Titre: Title:	Conception et optimisation par algorithme évolutif d'un neuro-contrôleur à pulsations embarqué sur un robot suiveur
Auteur: Author:	Jeffrey Dungen
Date:	2007
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Dungen, J. (2007). Conception et optimisation par algorithme évolutif d'un neuro-contrôleur à pulsations embarqué sur un robot suiveur [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7974/

UNIVERSITÉ DE MONTRÉAL

CONCEPTION ET OPTIMISATION PAR ALGORITHME ÉVOLUTIF D'UN NEURO-CONTRÔLEUR À PULSATIONS EMBARQUÉ SUR UN ROBOT SUIVEUR

JEFFREY DUNGEN
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(INTELLIGENCE ARTIFICIELLE)
AVRIL 2007

Your file Votre référence
ISBN: 978-0-494-29230-3
Our file Notre référence
ISBN: 978-0-494-29230-3

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

CONCEPTION ET OPTIMISATION PAR ALGORITHME ÉVOLUTIF D'UN NEURO-CONTRÔLEUR À PULSATIONS EMBARQUÉ SUR UN ROBOT SUIVEUR

présenté par: DUNGEN Jeffrey

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M NERGUIZIAN Chahé, PhD, président

M BRAULT Jean-Jules, PhD, membre et directeur de recherche

M BOUDREAULT Yves, PhD, membre

Cette mémoire est dédié à ma mère. Désolé de ne pas l'avoir fini à temps Mummy.

REMERCIEMENTS

Merci à mon directeur Jean-Jules Brault.

Merci à ma famille, mes collègues et mes amis qui m'ont encouragé et poussé à écrire ce mémoire.

RÉSUMÉ

La plupart des implantations des réseaux de neurones artificiels (RNA) sont des logiciels informatiques qui obéissent à des calculs mathématiques. Les implantations sous forme matérielle sont majoritairement réalisées avec des circuits numériques. Les RNA à impulsions (RNAi) implémentés sur circuit ITGÉ analogique représentent pourtant une alternative très intéressante car ils sont très efficaces tant du point de vue temps de calcul que de l'énergie dissipée.

Une hybridation numérique et analogique à impulsions pourrait permettre de concevoir des RNA pouvant profiter des avantages de ces deux types de circuits. Dans ce travail, notre intention est de montrer que des circuits électroniques de RNAi peuvent être rapidement conçus et optimisés pour une application embarquée. Nous avons muni un simple robot simulé d'un cerveau primitif de quatre neurones à impulsions pour qu'il puisse remplir son rôle qui consiste à suivre le plus près possible une cible libre se déplaçant dans un environnement borné.

Nous avons développé les outils nécessaires pour faire du prototypage rapide. Un premier logiciel simule le comportement de différents réseaux de neurones à base de transistors et de condensateurs à partir de simulations Spice. Un deuxième logiciel optimise les performances de ces derniers à l'aide d'un algorithme évolutif. Ces outils sont maintenant disponibles sur internet (sous forme d'application Java Web Start) pour que d'autres chercheurs puissent concevoir des RNAi à d'autres fins que celle visée par ces travaux.

L'optimisation d'un RNAi, même petit, est une opération plutôt complexe étant donné la non-linéarité du comportement des circuits électroniques analogiques, des effets imprévisibles des pulsations sur le circuit à rétroaction, et la dynamique robot/cible/environnement. Notre algorithme évolutif permet de réaliser sans difficulté des gains de performance à chaque génération jusqu'à ce que le neurocontrôleur (le RNAi) embarqué sur le robot affiche une performance supérieure

à celle d'un contrôleur humain entraîné. Sachant que le circuit est constitué que de quatre neurones regroupés en deux paires conpétitives (une centaine de transistors en tout), nous avons proposé une approche pour la réalisation d'une puce reconfigurable comprenant des circuits de RNAi à paramètres reconfigurables.

ABSTRACT

The vast majority of artificial neural networks (ANN) are implemented in computer software as mathematical models. Material implementations of said networks typically use digital circuits. Pulsing ANNs implemented in analog VLSI represent an interesting alternative thanks to their efficiency in both calculation speed and energy dissipation.

A pulsing digital and analog hybrid allows the implementation of ANNs profiting from the benefits of both of these types of circuits. In this work, we intend to show that pulsing ANNs can be rapidly developed and optimised for an embedded application. We have equipped a simple simulated robot with a primitive brain comprised of four pulsing neurons, which enables it to successfully follow a randomly moving target in a closed environment.

We have developed the tools required for rapid prototyping. A software application simulates the behaviour of arbitrary pulsing ANNs based on Spice simulations. Another software application optimises the performance of the simulated networks by using an evolutionary algorithm. These tools are currently available on the internet (as Java Web Start applications) so that other researchers may develop pulsing ANNs for use related to their field of study.

The optimisation of a pulsing ANN, even one of small scale, is a complex operation given the non-linear behaviour of the analogue electronics, the unpredictable effect of pulse feedback and the dynamics of the robot, target and environment. Our evolutionary algorithm produces consistent gains in the performance of subsequent generations of robots, resulting in a neurocontroller (the pulsing ANN) which allows a robot to achieve a performance superior to a human controller. Given that the circuit consists of only four neurons grouped in competitive pairs (one hundred transistors in all), we propose the development of a versatile microchip consisting of pulsing ANN circuits with reconfigurable parameters.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES NOTATIONS ET DES SYMBOLES

LISTE DES ACRONYMES

CE	Contrôleur enfant
CP	Contrôleur parent
IPÉ	Intégration à petite échelle
ITGÉa	Intégration à très grande échelle analogique
LGPT	Le-gagnant-prend-tout
RdN	Réseau de neurones
TProx	Senseur de proximité de la cible
T+40	Senseur directionnel de la cible, orienté à 40° de rotation à droite de l'avant du robot
T+120	Senseur directionnel de la cible, orienté à 120° de rotation à droite de l'avant du robot
T-40	Senseur directionnel de la cible, orienté à 40° de rotation à gauche de l'avant du robot
T-120	Senseur directionnel de la cible, orienté à 120° de rotation à gauche de l'avant du robot

LISTE DES ANNEXES

INTRODUCTION

Depuis l'enfance, j'ai toujours été attiré par le mystère qui entoure le fonctionnement du cerveau, et la façon dont il pense et qu'il traite l'information. Quand on est jeune, on se contente d'explications incroyables sur tout autour de nous car tout a l'air si compliqué. On cherche le petit bonhomme qui allume la lumière du réfrigérateur lorsqu'on l'ouvre. Mais, l'âge, l'expérience et l'éducation augmentent notre compréhension du monde, souvent au détriment de l'imagination. Plus on poursuit notre chemin d'étudiant en génie, plus on réalise que pratiquement toute la technologie est basée sur des phénomènes très simples. Même les plus puissants ordinateurs numériques au monde n'exécutent séquentiellement qu'un répertoire limité d'instructions, mais à très, très haute vitesse. Cependant, même après toutes nos études, le cerveau demeure toujours compliqué et même mystérieux, comme durant notre jeunesse. Évidemment, maintenant je sais c'est quoi un neurone, une synapse, des interconnections et des neurotransmetteurs, et j'ai une bonne idée comment fonctionnent chacun de ces derniers, mais je suis toujours incapable d'expliquer comment mon cerveau me permet d'écrire cette phrase. Par contre, je suis capable d'expliquer en détails comment l'ordinateur devant moi l'affiche sur l'écran. Le cerveau ne provient pas de notre technologie, et la science est toujours loin de le maîtriser.

Une de mes ambitions est d'arriver un jour à mieux comprendre le fonctionnement du cerveau animal. Comme ingénieur électrique, je suis mal équipé pour poursuivre cette ambition par l'approche biologique ou psychologique. Je suis pourtant habile à concevoir et analyser des machines électroniques. Comme un enfant construit avec des Lego^{TM}un modèle de ce qu'il voit dans le monde, mes habilités me permettent de modéliser ce que je comprends avec des circuits électroniques. C'est à dire que pour m'aider à mieux comprendre le cerveau, je pourrai en construire un modèle

très simple à l'aide de circuits électroniques, puis le tester et analyser certains de ses comportements. L'avantage de cette approche est qu'il est beaucoup plus facile d'analyser un cerveau artificiel qu'un cerveau biologique puisqu'on travaille avec notre propre technologie.

Il y a au moins une autre avantage de cette approche. À l'assemblée plénière de la conférence IJCNN 2005, Carver Mead a demandé à l'audience pourquoi après toutes les avances en informatique dans les 45 derniers ans, on n'était toujours pas capable de concurrencer même une simple mouche. J'ai eu la chance de lui demander si c'était peut-être un peu prétentieux de la part des humains de penser qu'ils seront capables, en quelques décennies, de surpasser en finesse des milliards d'années d'évolution. Un point critique de sa réponse : on n'a accès qu'aux fruits de cette évolution - les milliards d'années d'essais, d'erreurs et de cerveaux manqués sont perdus et demeuront toujours largement inconnus. C'est à dire qu'en étudiant le cerveau, on ne fait que de l'ingénierie inverse - et sur une machine tellement complèxe, on risque de perdre beaucoup d'intuition en procédant ainsi. Par contre, en concevant des cerveaux artificiels avec des circuits électroniques, on prend conscience des justifications des caractéristiques d'un design.

J'utilise ces deux avantages comme justification pour mon approche. Dans ma recherche, j'essai de concevoir des cerveaux artificiels conçus à partir de circuits électroniques simulés, et de les faire optimiser par évolution. Plus spécifiquement, j'utilise ces cerveaux artificiels, inspirés par la biologie, pour contrôler un robot simulé. Évidemment, ceci n'est pas révolutionaire ; il y a énormement d'individus qui font du contrôle robotique par des moyens plus ou moins inspirés biologiquement (voir, par exemple (Floreanu et al., 2006)). Par contre, ce qui est moins commun dans ma recherche est le fait de proposer l'utilisation *exclusive* de circuits de neurones à impulsions pour le contrôle *complet* d'un robot simple, et éventuellement facile à construire pour dépasser l'étape de la simulation informatif.

Ce qui suit est une petite historique de mon cheminement pour amener le lecteur au point de départ de ce mémoire. Quand j'ai commencé ma maîtrise à l'École Polytechnique de Montréal, j'avais comme but d'utiliser des contrôleurs analogiques pour donner aux robots un comportement plus dynamique, plus réel. À cette époque, j'avais beaucoup plus d'expérience avec des contrôleurs analogiques que des réseaux de neurones. Mais beaucoup de lecture, d'exploration et un cours en microélectronique analogique et mixte m'ont fait découvrir des circuits de neurones à impulsions de type *leaky integrate-and-fire*, synthétisables en ITGÉ analogique. Ceux-ci m'ont captivés, puisque leur principe de fonctionnement est semblable à celui des neurones biologiques, et les circuits sont très élégants. Comme projet pour ce cours, nous les avons organisé en réseaux simples, à poids fixes, avec un circuit pour régler automatiquement le gain (Dungen et Gherbi, 2003).

La session qui suivait, j'ai pris un cours en réseaux de neurones qui m'a fait découvrir les cartes auto-organisatrices de Kohonen (Kohonen, 1982). Comme projet pour ce cours, j'ai présenté des cartes auto-organisatrices conçus à partir de neurones à impulsions, avec des circuits pour implémenter un algorithme d'adaptation des poids synaptiques. En simulation, une carte unidimensionnelle de ces neurones était capable de s'auto-organiser pour approximer la distribution et la topographie d'une amplitude relative sur ses deux entrées. Après le cours, j'ai décidé de rendre mon travail plus accessible et plus visible à mes collègues et ma famille. J'ai développé en Java une version graphique et en-ligne de la carte auto-organisatrice de neurones à impulsions (Dungen, 2004) présentée à la figure 1.

Ce projet a donné de bons résultats, mais je trouvais que les circuits d'adaptation manquaient d'élégance. J'ai cherché d'autres moyens pour faire de l'auto-organisation ou de l'apprentissage dans des circuits de neurones à impulsions, mais aucun circuit n'était aussi élégant que les circuits neuronnaux eux-mêmes. Ceci m'a fait réaliser que l'adaptation en temps-réel n'était pas une nécessité, et que

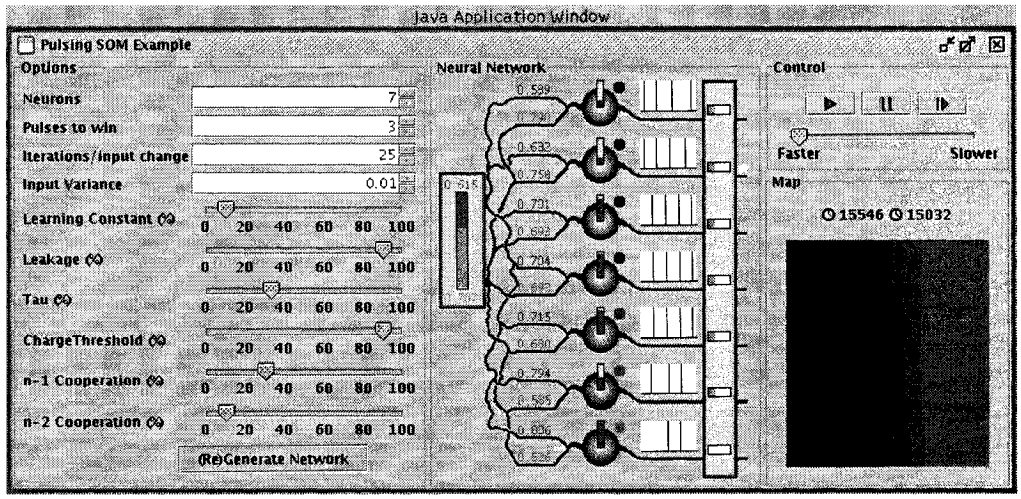

FIG. 1 Application en-ligne de la carte auto-organisatrice

c'était probablement un but trop ambitieux. Dans le royaume animal, les systèmes nerveux les plus simples s'adaptent plutôt de génération à génération que pendant la durée de vie d'un individu. Un but plus réaliste serait donc de concevoir des cerveaux artificiels simples à paramètres fixes, et de les optimiser par évolution.

Mon premier pas dans cette direction était d'avancer ce projet avec un réseau à poids fixes dont les modèles des circuits provenaient de simulations Spice (dans le projet précédent, ces modèles étaient très simplifiés). Toujours en Java, j'ai développé une application graphique et en-ligne où une couche de neurones simulée était capable de caractériser l'amplitude relative entre deux entrées non-normalisées (pour la carte auto-organisatrice, je les normalisais), présentée à la figure 2 (Dungen, 2004). Confiant qu'avec les modèles de circuits plus raffinés on obtenait toujours les mêmes résultats positifs, j'ai developpé une application où les neurones contrôlaient le comportement d'un simple robot selon les sons entendus par ses oreilles artificielles, présentée à la figure 3 (Dungen, 2004). Cette fois-ci, le réseau comprenait deux couches. Dans la première, des groupes de neurones caractérisaient la fréquence, l'amplitude et la direction du son tandis que dans la deuxième couche, des groupes de neurones décidaient la direction et la vitesse du

robot. Avec des paramètres optimisées par essai-erreur, le robot se dirigeait vers une source de bruit à haute fréquence (qui représente le plaisir), et se cachait d'une source de bruit à basse fréquence (qui représente la peur). Le robot avait des mouvements érratiques et réagissait lentement et parfois même pas du tout, ce qui lui donnait un certain individualité. J'étais fortement impressionné par les capacités de son réseau de quatorze neurones simples et efficaces. Par contre, il n'était pas facile d'améliorer le réseau, parce que chaque changement de paramètres nécessitait une recompilation et redémarrage de l'application. La suite logique c'était de développer une application qui permettrait la construction et les tests de réseaux arbitraires de ces neurones à impulsions avec des paramètres ajustables. Ceci nous amène aux buts de mon mémoire.

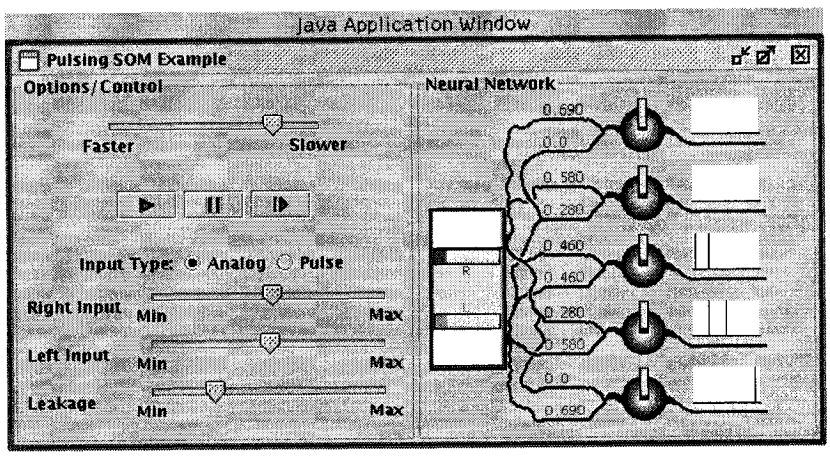

FIG. 2 Application en-ligne de la carte capable de classifier l'amplitude relative entre deux entrées non-normalisées

Mes buts sont de montrer qu'un simple cerveau artificiel composé de circuits de neurones à impulsions est capable de contrôler un simple robot de manière intéressante et de pouvoir l'évoluer, de génération en génération, pour donner au robot une meilleure performance sur une tâche donnée. En d'autres mots, ce mémoire est un preuve du concept qu'un réseau de neurones à impulsions efficace et réalisable en matériel puisse servir comme contrôleur embarqué.

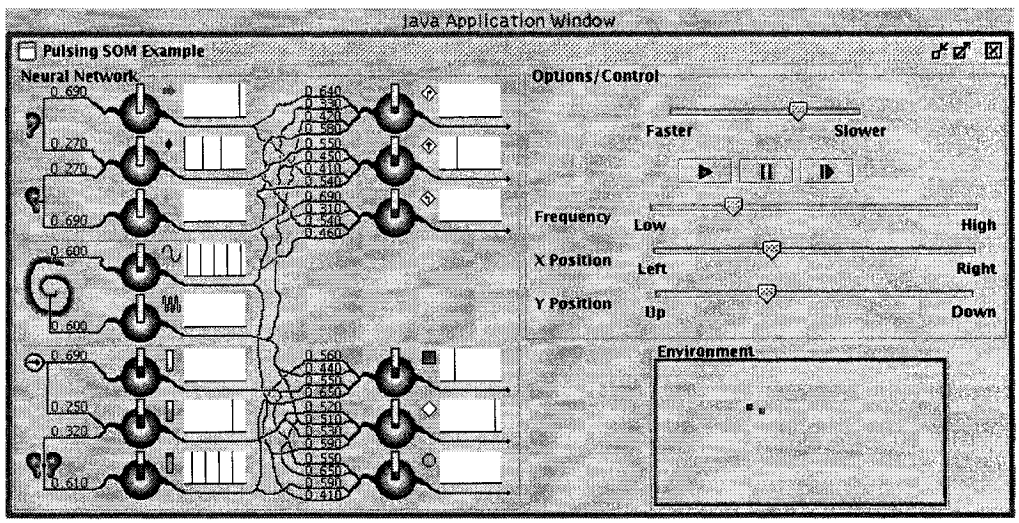

FIG. 3 Application en-ligne de contrôleur de robot simple

Il existe, à ma connaissance, aucune plateforme librement disponible pour la conception et la simulation de circuits de neurones à impulsions de type, *leaky integrate-and-fire*. Mon premier travail est donc de développer une telle plateforme. De suite, c'est de simuler, grâce à cette plateforme, le contrôle par ces neurones d'un robot simple dans un environnement limité. Ensuite, c'est de développer une plateforme d'optimisation par évolution et d'en servir pour générer un contrôleur de robot performant.

En résumé, mes objectifs sont de mettre en place l'infrastructure pour la conception, simulation et optimisation de neurones à impulsions de type *leaky integrate-and-fire* pour le développement d'un contrôleur de robot primaire, facilement réalisable en matériel, et ensuite, d'en servir pour montrer que ces circuits neuronnaux efficaces puissent justement contrôler ce robot de manière intéressante. Au premier et au deuxième chapitre, on présente les circuits de neurones à impulsions et le logiciel qui simule leur comportement. Au troisième et au quatrième chapitre, on présente le robot qu'on vise contrôler, puis on développe un réseau de neurones simulé qui lui donne un comportement souhaitable. Finalement, on conclut en résumant les

avantages et capacités de la technologie et les avenues de recherche futures.

CHAPITRE 1

NEURONES À PULSATIONS EN ITGÉ ANALOGIQUE

1.1 Le neurone biologique

Le cerveau animal est composé de neurones massivement interconnectés entre eux. La figure 1.1 représente un modèle simplifié d'un neurone biologique. Le neurone pondère ses multiples entrées pour générer une sortie sur son axone unique. Les entrées d'un neurone sont les sorties d'autres neurones : les synapses d'un neurone sont le point de connexion aux axones des autres. Chaque synapse transforme le signal d'entrée selon une pondération unique, ce qui représente de la multiplication. Les signaux tranformés sont acheminés vers le corps du neurone, le soma, par des dendrites. Le soma fait une sommation de toutes ses entrées pondérées, et détermine la sortie selon une fonction d'activation. Quand le seuil de cette fonction est dépassé, le soma sort une impulsion qui est transmise sur l'axone et envoyée vers d'autres neurones.

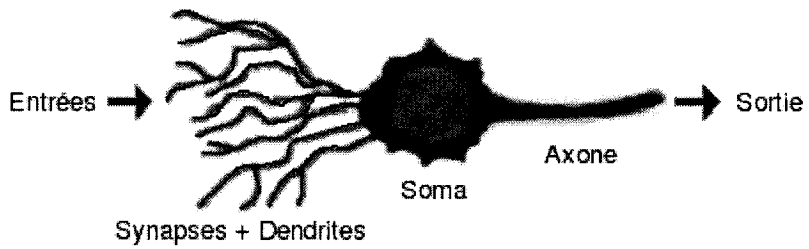

FIG. 1.1 Modèle simplifié d'un neurone biologique

Cette description d'un neurone ne représente qu'une simplification du cas général, mais elle suffit pour présenter la nomenclature et pour identifier deux propriétés

importantes : les synapses font la multiplication, et le soma fait une sommation suivie d'une fonction d'activation.

1.2 L'état de l'art en conception de neurones à impulsions en ITGÉ analogique

Il est intéressant de rappeler que l'ITGÉ est beaucoup plus récent que les premiers réseaux de neurones artificiels. Ces derniers datent des années 1950 tandis que ce n'est qu'en 1979 que Carver Mead et Lynn Conway ont écrit *Introduction to VLSI Systems*, un des textes pionniers de l'ITGÉ, souvent référencé même aujourd'hui. À peu près une décennie plus tard, Mead a écrit un autre texte pionnier, cette fois-ci concernant les systèmes neuronaux en ITGÉ analogique (Mead (Ed.) et Ismail (Ed.), 1989). À partir de ce point, d'autres chercheurs présentaient les premiers systèmes neuromorphiques : des circuits, inspirés par le système nerveux biologique, implémentés en ITGÉ analogique, par exemple (Lazzaro et Mead, 1989), (Delbrück, 1991), (Maass et Turan, 1994) et (Indiveri, 2001). Aujourd'hui il existe plusieurs branches de recherche tels que l'optimisation des circuits et la conception de systèmes neuromorphiques à grande échelle. Pour ce mémoire, on s'intéresse exclusivement aux neurones à impulsions de type *leaky integrate-and-fire* en ITGÉ analogique, ce qui ne représente qu'un petit sous-groupe de la recherche courante.

Selon Gerstner et Kistler, le neurone à impulsions de type *leaky integrate-and-fire* est probablement le plus répandu parmi tous les modèles formels de neurones à impulsions (Gerstner et Kistler, 2002). C'est aussi parmi les plus simples. Les circuits de neurones utilisés dans ce mémoire sont basés sur ceux présentés aux étudiants du cours *Computation in Neuromophic Analog VLSI Systems* à l'Institut de neuroinformatique à l'université de Zürich (Indiveri, 2003). Il existe une variété de circuits optimisés pour la compacité, le réalisme biologique ou la faible consommation de

puissance (pour des exemples, voir l'annexe I). De manière générale, ces qualités sont mutuellement exclusives. Les circuits utilisés dans ce mémoire sont orientés vers la compacité. Des circuits semblables sont utilisés dans la recherche courante, justement à cause de leur simplicité. Deux exemples qui servent à l'audition et au contrôle de la locomotion sont, respectivement, (Glover et al., 2002) et (Lewis et al., 2000).

Dans les sections suivantes, on présente les circuits utilisés dans ce mémoire, qui sont réalisables dans un processus CMOSp de 0.18 microns, ce qui représentait l'état de l'art en 2003, au début de cette recherche. Les dimensions des transistors dans les circuits synaptiques et somatiques sont préscrites par l'auteur et ne sont pas optimisées afin d'assurer le fonctionnement des circuits. La tension d'alimentation (V_{DD}) est 1.8V. On présente les circuits de bas à haut niveau, c'est à dire les circuits synaptiques et somatiques en premier, suivis par le circuit neuronal qui comprend ces derniers, et finalement le circuit le-gaganant-prend-tout qui regroupe plusieurs neurones.

1.3 Circuit synaptique

La synapse représente le point d'entrée d'un neurone. Son circuit, présenté à la figure 1.2, comprend quatre transistors et un condensateur. L'entrée principale de la synapse est la tension présynaptique (V_{entree}). Cette entrée peut être soit des impulsions, quand, par exemple, c'est la sortie d'un neurone, ou une tension analogique variable, quand, par exemple, c'est la sortie d'un senseur analogique. Ces deux types d'entrée amènent des comportements distincts, donc les paramètres synaptiques doivent en tenir compte. L'entrée secondaire de la synapse est la tension de poids (V_{poids}), qui détermine l'effet de l'entrée principale sur la sortie. L'entrée tertiaire est la tension de gain (V_{gain}) qui peut simplement être la tension d'alimen-

TAB. 1.1 Paramètres synaptiques

Paramètre	*Description*	*Valeur*
V_{entree}	Impulsions ou tensions analogiques. Normalement le seul paramètre dynamique du circuit.	0.0 à 1.8V
V_{poids}	Détermine l'effet de l'entrée sur la sortie. Ajustable pour donner le comportement désiré.	0.0 à 0.7V
V_{gain}	Détermine l'effet de l'entrée et le poids sur la sortie.	1.75V
C	Détermine la durée de l'effet d'une entrée transitoire sur la sortie.	8.0 nF
I_{sortie}	Courant de sortie = f(V_{entree}, V_{poids}, V_{gain}, C).	0.0 à 4.0uA

tation, ou une tension plus basse, justement pour augmenter le gain de la fonction de transfert. L'unique sortie est un courant qui dépend de ces trois entrées. Les paramètres du circuit, choisis par l'auteur selon les contraintes de la technologie, sont décrits dans la table 1.1.

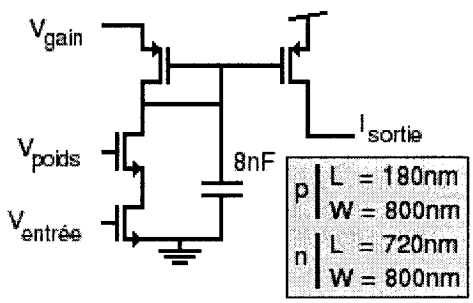

FIG. 1.2 Circuit d'une synapse

1.3.1 Principe de fonctionnement

Globalement, le circuit fonctionne ainsi. Le courant de sortie est inversement lié à la tension sur le condensateur. Alors, pour diminuer la sortie, il suffit de charger

le condensateur, et pour l'augmenter, il suffit de le décharger. Ceci est effectué par les trois transistors à gauche dans le schéma du circuit. Le pMOS sert à le charger, alors que les deux nMOS permettent de le décharger. Plus V_{gain} est bas, plus il-y-a de courant qui charge le condensateur. Plus V_{entree} et V_{poids} sont élevées, plus le condensateur se décharge. La relation entre V_{entree} et V_{poids} pour déterminer la sortie est décrite à la figure 1.3. La capacité du condensateur n'affecte que la durée de chargement et de déchargement. Plus le condensateur a une grande capacité, plus long sera l'effet d'une impulsion à l'entrée sur la sortie. Cet effet est peu prononcé lorsque l'entrée est une tension analogique qui varie lentement, mais pour des impulsions, ça affecte la durée d'activité. La capacité de 8nF fait que les effets d'une impulsion ne durent que quelques millisecondes, ce qui simplifit les modèles de simulation (voir section 2.1.1.2). Le comportement de la synapse face à une impulsion est présenté aux figures 1.4 et 1.5. La forme d'une impulsion est présentée à la figure 1.7. Toutes les simulations ont été réalisées en Spice (voir annexe II pour les fichiers source).

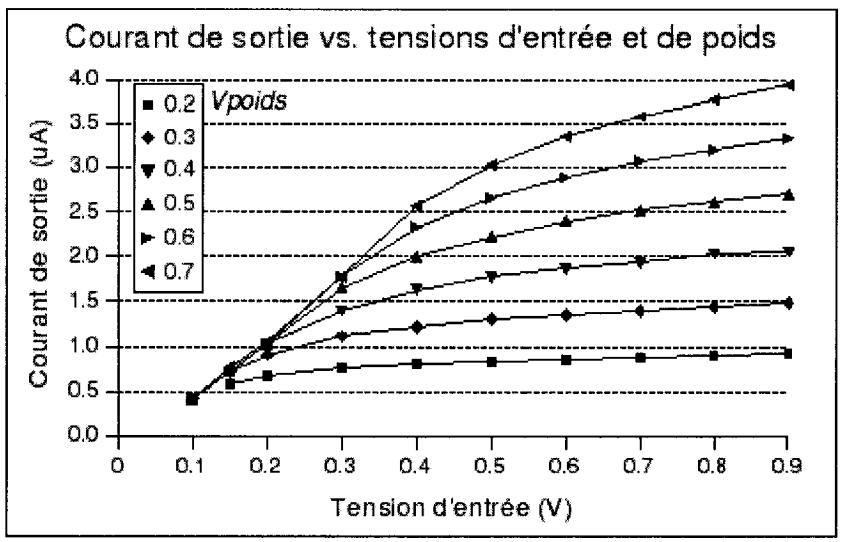

FIG. 1.3 Circuit synaptique : effet de l'entrée et le poids sur la sortie

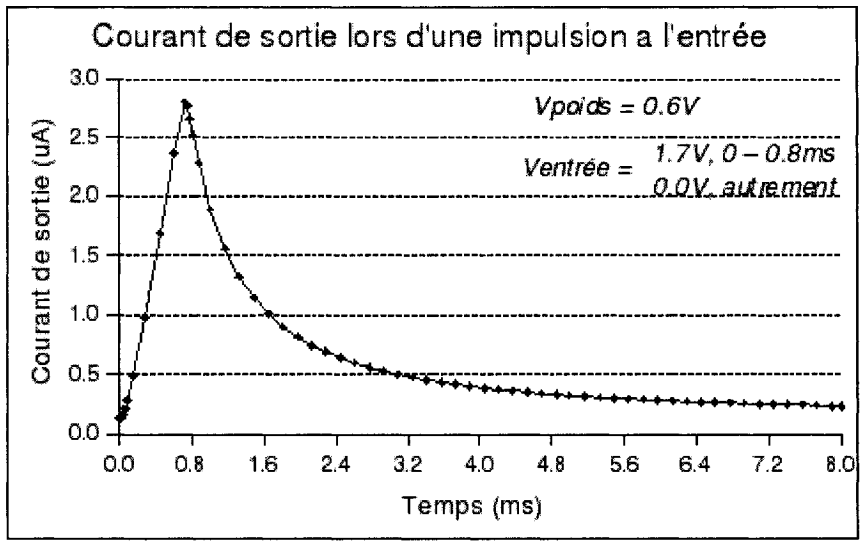

FIG. 1.4 Circuit synaptique : courant de sortie après une impulsion

1.4 Circuit somatique

Le soma représente le générateur d'impulsions du neurone. Son circuit, présenté à la figure 1.6, comprend (au moins) sept transistors et deux condensateurs. L'entrée principale du soma est le courant synaptique (I_{entree}), ce qui représente, selon le loi de courant de Kirchoff, la somme des courants de sortie de toutes les synapses qui y sont branchés (équation 1.1). L'entrée $V_{epaisseurd'impulsion}$ est une tension fixe qui détermine justement l'épaisseur des impulsions de sortie. L'entrée $V_{fuite/reset}$ correspond plutôt à plusieurs entrées (donc plusieurs transistors). Le soma a toujours une fuite, mais il peut avoir soit aucune ou plusieurs entrées *reset*. La fuite et les *reset* représentent chacun un transistor en parallèle avec les autres. La fuite est une tension fixe qui détermine le courant de déchargement du réservoir somatique (c'est à dire la charge sur le condensateur de 8nF). Le ou les entrées *reset* acceptent des impulsions qui peuvent chacune décharger complètement ce même réservoir. L'unique sortie émet une impulsion lorsque la charge dans le réservoir dépasse un seuil, sinon la sortie est nulle. La forme d'une impulsion est présentée

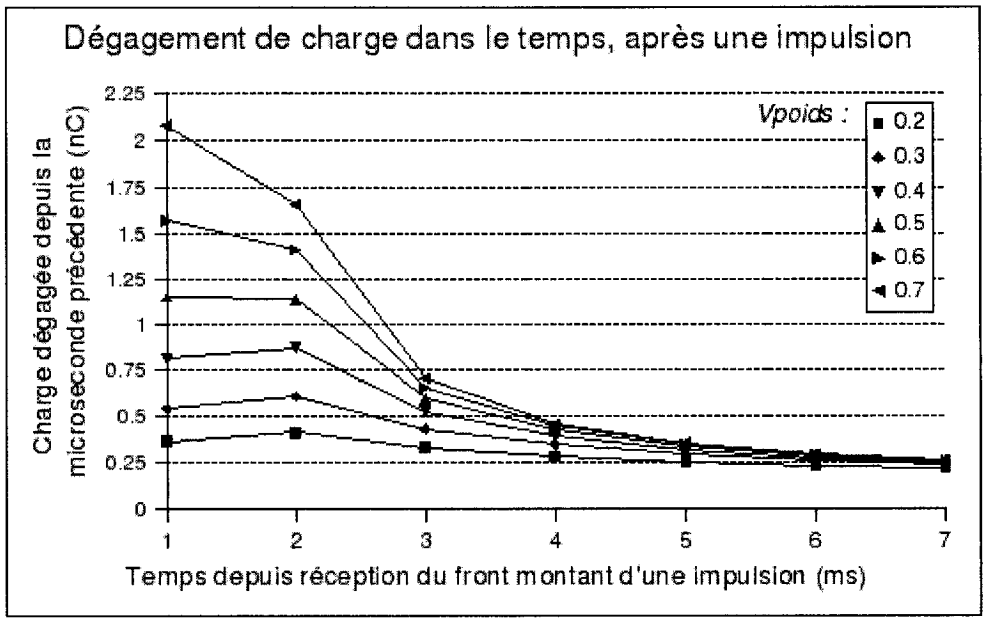

FIG. 1.5 Circuit synaptique : charge dégagée chaque milliseconde après une impulsion

à la figure 1.7, et les paramètres du circuit somatique sont décrits à la table 1.2.

$$I_{entree} = \sum I_{synaptique} \tag{1.1}$$

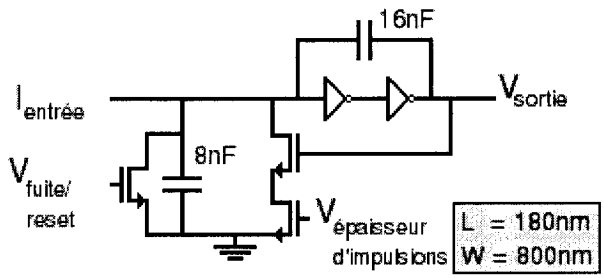

FIG. 1.6 Circuit d'un soma

TAB. 1.2 Paramètres somatiques

Paramètre	*Description*	*Valeur*
I_{entree}	Courant qui charge le réservoir et détermine la fréquence d'émission d'impulsions.	0 à des dizaines de uA
V_{fuite}	Détermine le courant de fuite qui, à son tour, détermine le courant minimum á l'entrée pour l'émission d'impulsions.	0.0V et plus
V_{reset}	Décharge le condensateur quand une impulsion y est appliquée. Ramène le soma au potentiel de répos.	0.0 ou env. 1.7V
$V_{epaisseurdepulsations}$	Détermine la durée des impulsions à la sortie.	0.9V
V_{sortie}	Émission d'une impulsion = f(I_{entree}, V_{fuite}, V_{reset}, t).	0.0 ou env. 1.7V

1.4.1 Principe de fonctionnement

Globalement, le circuit du soma fonctionne ainsi. Le courant d'entrée charge continuellement les condensateurs en même temps qu'ils sont déchargés par le transistor de fuite (Équation 1.2). Quand le courant d'entrée dépasse celui de fuite, les condensateurs chargent jusqu'à ce que la tension à l'entrée du premier inverseur dépasse le seuil de logique-1 (0.9V, imposé par l'inverseur). À ce moment, le premier inverseur sort logique-0, et de suite le deuxième sort logique-1. Ceci donne le front montant d'une impulsion à la sortie. Durant l'impulsion, la sortie est haute-tension (logique-1), et alors le condensateur est déchargé via la paire de transistors incluant celui gérant l'épaisseur d'impulsions. Ce déchargement diminue la tension devant le premier inverseur, et quand celle-ci tombe en dessous du seuil de logique-1, les inverseurs retournent à leur état de repos en donnant le front descendant de l'impulsion à la sortie. Le condensateur 8nF détermine combien d'activité sur l'entrée est nécessaire avant qu'une impulsion soit émise. La relation entre courant d'entrée

et fréquence d'impulsion est présentée à la figure 1.8. Cette relation est linéaire, ce qui indique que 23.3nC de charge doivent s'accumuler avant qu'une impulsion soit émise (équation 1.3). Le condensateur 16nF affecte la forme de l'impulsion, ce qui est présentée à la figure 1.7. Toutes les simulations ont été réalisés en Spice (voir annexe II pour les fichiers source).

$$I_{chargementDeReservoir} = I_{entree} - I_{fuite} \tag{1.2}$$

$$C_{seuil} = 23.3nC \tag{1.3}$$

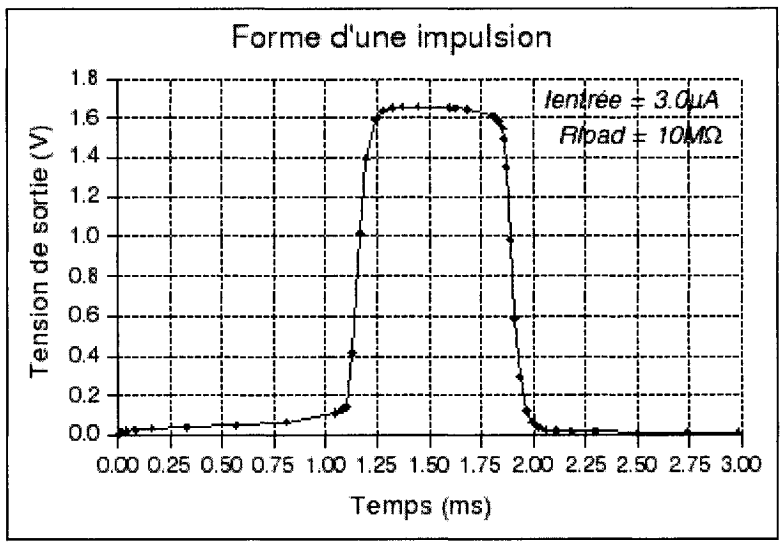

FIG. 1.7 Circuit somatique : forme d'une impulsion

1.5 Circuit neuronal

Le neurone représente l'unité de base d'un réseau de neurones. Son circuit, présenté à la figure 1.9 comprend une ou plusieurs synapses et un seul soma. Le neurone a deux entrées par synapse, une entrée de fuite, une entrée d'épaisseur d'impulsions et autant d'entrées de reset que nécessaire. Il n'y a qu'une seule sortie (impulsion ou

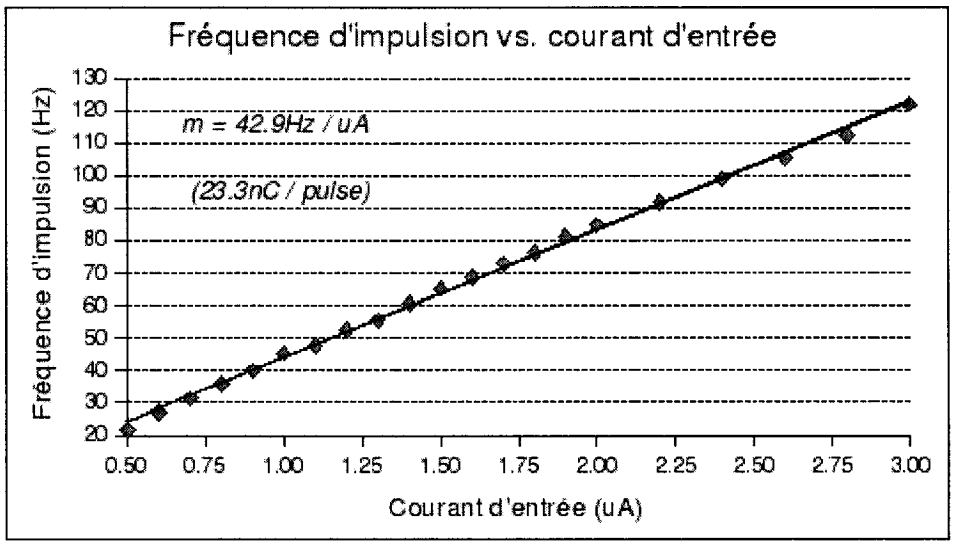

FIG. 1.8 Circuit somatique : relation entre courant d'entrée et fréquence d'impulsion

non). Les circuits synaptiques et somatiques et leurs entrées et sorties sont décrits dans les sections 1.3 et 1.4 respectivement.

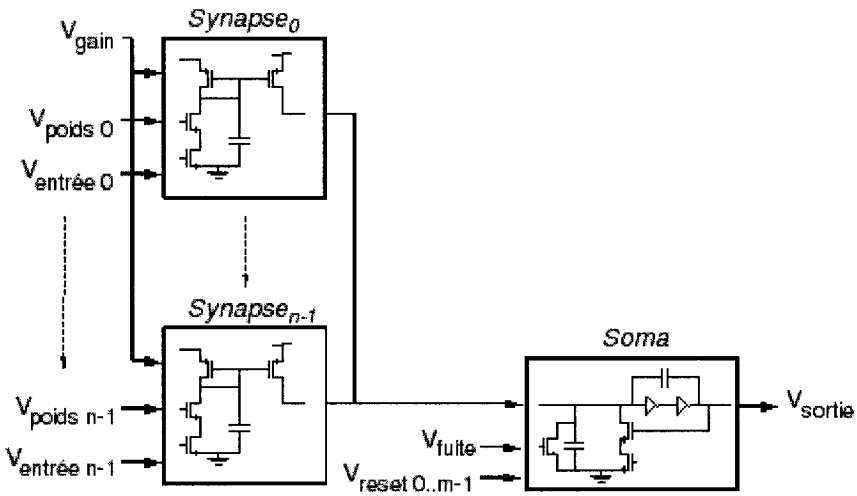

FIG. 1.9 Circuit d'un neurone

1.5.1 Principe de fonctionnement

Le neurone se résume ici à un soma ayant un courant d'entrée (I_{entree}) variable en fonction des synapses et de leurs entrées. Chaque synapse produit un courant de sortie distinct, en fonction de son entrée et de son poids, et la somme de ces courants représente l'entrée du soma (équation 1.1). Le soma intègre ceci, moins le courant de fuite, et émet une impulsion à chaque fois que la charge accumulée dépasse le seuil de 23.3nC. Par contre, une impulsion appliquée à n'importe quelle entrée *reset* remet presque instantanément l'intégration à zéro. La fréquence d'émission d'impulsions est constante pour une I_{entree} fixe, tel qu'indiquée à la figure 1.8.

1.6 Circuit le-gagnant-prend-tout (LGPT)

Un regroupement le-gagnant-prend-tout (LGPT) est un ensemble de neurones dont les sorties sont mutuellement exclusives. C'est à dire que lorsque un des neurones du groupe émet une impulsion, les autres subissent un *reset* pour que la compétition pour émettre une impulsion recommence à nouveau. Ce comportement est facilement effectué en branchant la sortie de chaque neurone avec une entrée *reset* de tous les autres neurones dans le groupe. De plus, il est possible de configurer dynamiquement et numériquement les regroupements en mettant une porte ET sur le chemin entre chaque sortie de neurone et chaque entrée *reset* des autres, tel que présenté à la figure 1.10. Évidemment, pour qu'un neurone puisse *resetter* un autre, il faut que l'autre entrée de la porte ET qui les rejoigne soit logique-1. Les *Gr.#/#* de la figure 1.10 représentent ces entrées. Le premier chiffre indique le neurone ciblé pour le *reset* et le deuxième chiffre indique le neurone qui appuye le *reset*. Par exemple, quand Gr.0/1 = logique-1, une impulsion à la sortie du neurone 1 effectuera un *reset* du neurone 0. Les regroupements LGPT peuvent donc être

configurés dynamiquement et numériquement. Selon nos connaissances, les circuits présentés qui implémentent cette technique sont nouveaux.

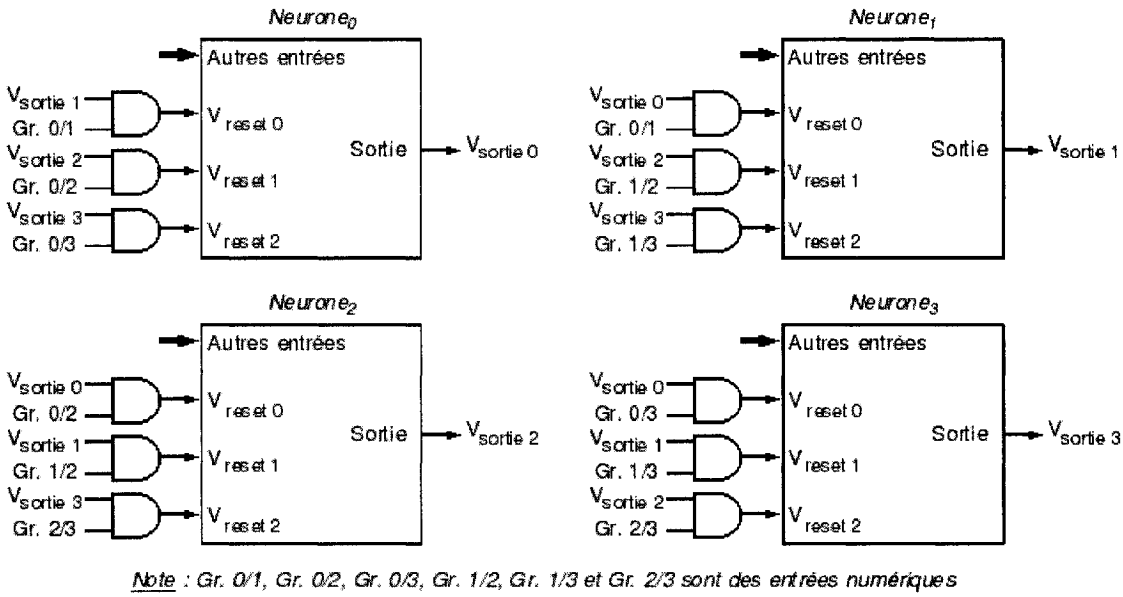

FIG. 1.10 Circuit le-gagnant-prend-tout (LGPT)

1.7 Conclusion

Dans ce chapitre, on a présenté les circuits de neurones à pulses utilisés dans ce mémoire. Les comportements de ces circuits ont été obtenus par des simulations Spice. Ces derniers sont idéales pour étudier le comportement d'un seul neurone, synapse ou soma, mais quant à la simulation de réseaux de neurones, la simulation en Spice est surdétaillée et encombrante. Il faudrait plutôt un simulateur de comportement haut-niveau avec interface utilisateur graphique. Étant donné qu'un tel logiciel n'est pas librement disponible, on le développe nous même, comme présenté au chapitre suivant.

CHAPITRE 2

PLATEFORME DE DÉVELOPPEMENT ET SIMULATION

Dans ce chapitre on présente l'application de développement et de simulation de circuits de réseaux de neurones de type *leaky integrate-and-fire*, programmée en Java. L'application consiste en deux parties : d'une part des modèles des circuits qui simulent leur comportement, et d'autre part une interface utilisateur graphique permettant la conception rapide de réseaux de neurones et la visualisation de leur comportement en temps-réel. On présente en premier lieu les modèles de simulation utilisés, puis, par la suite, l'interface utilisateur graphique et ses fonctionnalités.

2.1 Modèles de simulation

Dans les sections 1.3, 1.4, 1.5 et 1.6, on présentait le comportement des circuits synaptiques, somatiques, neuronnaux et le-gagnant-prend-tout. Des modèles de leurs comportements sont utilisés pour les simuler dans l'application. C'est important que ces modèles soient réalistes, mais en même temps assez simplifiés pour que la simulation puisse se dérouler en temps-réel.

Pour simuler le comportement dans le temps, il faut d'abord choisir une résolution temporelle. Nous avons décidé d'utiliser une résolution temporelle de 1 milliseconde puisque cela correspond à la durée de l'évènement le plus court dans le système : une impulsion. C'est à dire que la simulation se déroule itération par itération, où chaque itération représente 1 milliseconde.

On présente maintenant les modèles des circuits synaptiques, somatiques, neuron-

naux et le-gagnant-prend-tout.

2.1.1 Modèle du circuit synaptique

L'application utilise en fait deux modèles pour le circuit synaptique. Un modèle simule le comportement d'une synapse lorsque l'entrée est une tension analogique, et l'autre lorsqu'il s'agit d'impulsions. Ce deuxième modèle est nécessaire pour représenter les effets temporelles qui n'ont lieu qu'avec des entrées transitoires. Dans le cas où l'entrée est une tension analogique, on peut ignorer ces effets à moins que l'entrée ne varie pas à des fréquences supérieures à quelques centaines de Hertz. Dans les deux modèles, la sortie est calculée comme étant un montant de charge dégagé durant l'itération (de 1 ms). On présente tout d'abord le modèle du circuit synaptique en mode analogique, puis, par la suite, le modèle du circuit synaptique en mode impulsion. Les deux utilisent des tables de conversion.

2.1.1.1 Modèle du circuit synaptique en mode analogique

Le modèle du circuit en mode analogique est basé sur les données des simulations en Spice, comme celles présentées à la figure 1.3. On suppose que l'entrée ne varie pas assez rapidement pour donner des effets à délai, et donc qu'il n'y a que deux paramètres qui affectent la sortie : V_{entree} et V_{poids}. La relation entre ces paramètres et la sortie est complexe, alors on le modélise avec une table de conversion au lieu d'une fonction de transfert. Ce modèle comprend des points de données connus (V_{entree}, V_{poids}, Charge/Itération) et utilise l'interpolation pour estimer la sortie quand les entrées se situent entre quatres points de données connues. Les données du modèle sont présentées à la table 2.1. La résolution des V_{entree} et V_{poids} est de 100 mV.

TAB. 2.1 Modèle synaptique analogique : charge dégagée par itération (nC)

	Tension à l'entrée									
V_{poids}	*0.0V*	*0.1V*	*0.2V*	*0.3V*	*0.4V*	*0.5V*	*0.6V*	*0.7V*	*0.8V*	*0.9V*
0.0V	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.1V	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.2V	0.00	0.00	0.68	0.77	0.82	0.84	0.86	0.89	0.91	0.93
0.3V	0.00	0.42	0.91	1.12	1.21	**1.30**	**1.35**	1.40	1.44	1.49
0.4V	0.00	0.42	1.03	1.40	1.63	**1.77**	**1.86**	1.93	2.03	2.05
0.5V	0.00	0.42	1.03	1.65	2.00	2.21	2.40	2.52	2.61	2.70
0.6V	0.00	0.42	1.03	1.77	2.33	2.66	2.89	3.08	3.19	3.33
0.7V	0.00	0.42	1.05	1.77	2.56	3.03	3.36	3.59	3.77	3.94

La méthode d'interpolation utilisée est une somme pondérée des quatres entrées les plus proches dans la table. Les pondérations sont fonction de la proximité relative à chacune d'elle. Ceci s'explique mieux par un exemple. Imaginons que l'on ait $V_{poids} = 0.36$V et $V_{entree} = 0.52$V. Les deux rangées de poids dans la table les plus proches de 0.36V sont 0.3V et 0.4V. Les deux colonnes d'entrées dans la table les plus proches sont 0.5V et 0.6V. Les quatres points sont alors ceux en caractères gras dans la table 2.1. On pondère linéairement, ce qui donne 40% de la pondération dans la rangée 0.3V et l'autre 60% dans la rangée 0.4V. Pour V_{entree}, on donne 80% de la pondération dans la colonne 0.5V et l'autre 20% dans la colonne 0.6V. La valeur interpolée est alors :

$$\begin{aligned} Charge &= (40\% \times 80\% \times 1.30nC) + (40\% \times 20\% \times 1.35nC) \\ &\quad +(60\% \times 80\% \times 1.77nC) + (60\% \times 20\% \times 1.86nC) \\ &= (32\% \times 1.30nC) + (8\% \times 1.35nC) + (48\% \times 1.77nC) + (12\% \times 1.86nC) \\ &= 1.60nC \end{aligned} \tag{2.1}$$

Notez que les pondérations somment toujours à 100%. À chaque itération, cette

TAB. 2.2 Modèle synaptique impulsion : charge dégagée par itération (nC)

	Nombre d'itérations depuis l'impulsion						
V_{poids}	*0*	*1*	*2*	*3*	*4*	*5*	*6*
0.0V	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.1V	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.2V	0.37	0.41	0.33	0.29	0.26	0.23	0.22
0.3V	0.54	0.61	0.43	0.35	0.29	0.26	0.24
0.4V	0.81	0.87	0.53	0.39	0.32	0.28	0.25
0.5V	1.15	1.14	0.60	0.42	0.34	0.29	0.26
0.6V	1.58	**1.42**	0.66	0.45	0.35	0.29	0.26
0.7V	2.08	**1.66**	0.71	0.46	0.35	0.30	0.26

méthode est appliquée pour calculer la charge dégagée à la sortie.

2.1.1.2 Modèle du circuit synaptique en mode impulsion

Le modèle du circuit en mode impulsion est basée sur les données des simulations en Spice, comme celles présentées à la figure 1.5. On suppose que les impulsions sont toujours très proches de la forme présentée à la figure 1.7, ce qui fait qu'il n'y a que deux paramètres qui affectent la sortie : V_{poids} et le temps depuis la dernière impulsion à l'entrée, mesuré en nombre d'itérations. Encore ici, la relation entre ces paramètres et la sortie est complexe, et donc on la modélise avec une table de conversion au lieu d'une fonction de transfert. Ce modèle utilise des points de données connus (V_{entree}, nombre d'itérations depuis la dernière impulsion à l'entrée, Charge/Itération) et utilise l'interpolation pour estimer la sortie lorsque les entrées se situent entre deux points de données connues. Les données comprises dans le modèle sont présentées à la table 2.2. La résolution des V_{poids} est de 100 mV.

La méthode d'interpolation utilisée est une somme pondérée des deux entrées les plus proches dans la table. Les pondérations sont fonction de la proximité relative

à chacune. Ceci s'explique mieux par une exemple. Imaginons que l'on ait $V_{poids} = 0.61\text{V}$ et qu'on est à la première itération suivant une impulsion. Les deux rangées de poids dans la table les plus proches de 0.61V sont 0.6V et 0.7V. La colonne 1 correspond à notre situation. Les deux points sont alors ceux en caractères gras dans la table 2.2. On pondère linéairement, ce qui donne 90% de la pondération dans la rangée 0.6V et l'autre 10% dans la rangée 0.7V. La valeur interpolée est alors :

$$\begin{aligned} Charge &= (90\% \times 1.42nC) + (10\% \times 1.66nC) \\ &= 1.44nC \end{aligned} \tag{2.2}$$

À chaque itération, cette méthode est appliquée pour calculer la charge dégagée à la sortie. Dans le cas rare où une deuxième impulsion est présente à l'entrée six itérations ou moins après la première, toute la charge restante qui correspond à la première est dite ajoutée à la charge correspondant à la "zéroième" itération de la deuxième impulsion. Aux itérations suivantes, les effets de la première impulsion sont ignorées. Ceci simplifie le modèle ainsi que son temps de calcul.

2.1.2 Modèle du circuit somatique

Le modèle du circuit somatique est basé sur son comportement simulé en Spice tel que décrit à la section 1.4. À chaque itération (i), la charge accumulée est calculée ainsi :

$$Charge_i = Charge_{i-1} + \sum \text{Charge des synapses} - Fuite \tag{2.3}$$

Ici la *Fuite* est mesurée en charge par itération, et il s'agit d'une constante propre à chaque neurone. La charge calculée est comparée au seuil nécéssaire pour générer une impulsion (23.3nC). Si la charge est égale ou supérieure à 23.3nC, une impulsion

est produite à la sortie. Finalement, si une impulsion est reçu à l'entrée *Reset* ou une impulsion est émise à la sortie, la charge accumulée est remise à zéro. La charge accumulée ne peut évidemment jamais descendre en bas de zéro.

2.1.3 Modèle du circuit le-gagnant-prend-tout (LGPT)

Le modèle du circuit LGPT fonctionne ainsi. À chaque itération, si un ou plusieurs neurones du regroupement émettent une impulsion, un *reset* est appliqué à tous les autres neurones du regroupement. Ce *reset* prend effet sur l'itération même. Il est possible d'avoir plus qu'un neurone qui émet une impulsion à la même itération dans le modèle du circuit LGPT.

2.2 Interface utilisateur graphique

L'application de développement et simulation comprend une interface utilisateur graphique pour faciliter la conception et la simulation des réseaux de neurones. Cette interface est présentée à la figure 2.1. Dans cette section, on décrit cette interface et ses fonctionnalités.

2.2.1 Plateforme

L'interface usager graphique est codée en Java, tout comme les modèles de circuits que l'on vient de présenter. Le langage de programmation Java est utilisé parce qu'il permet d'utiliser l'application sur différents types de systèmes embarqués et qu'il peut être démarré à partir d'une site web.

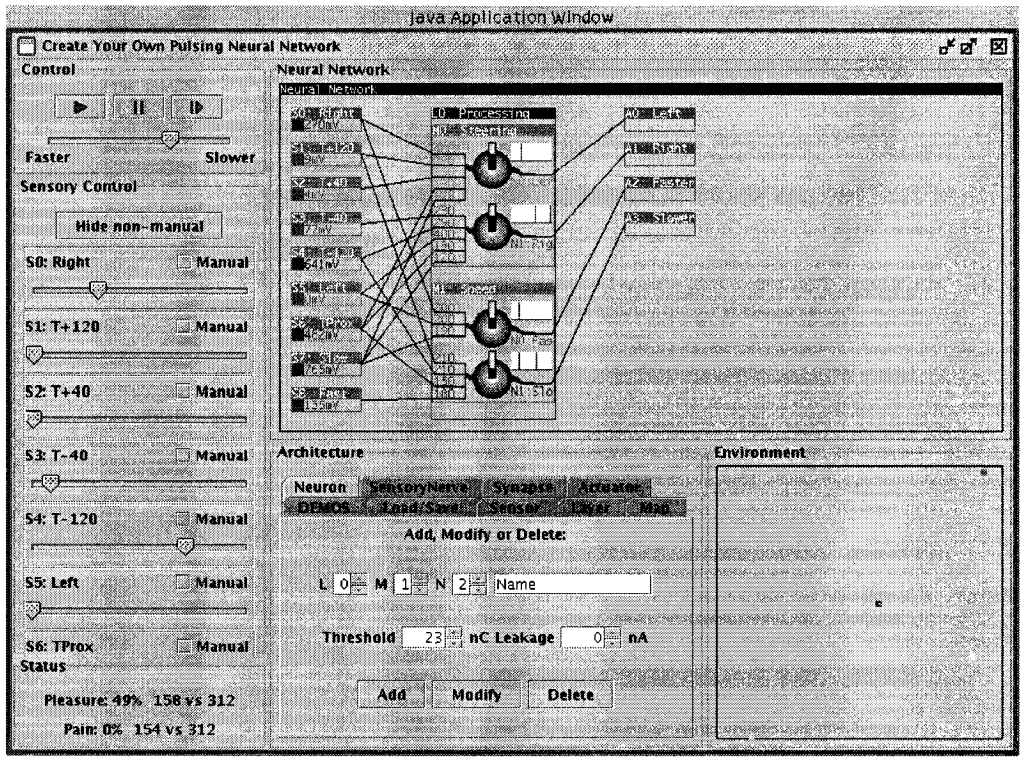

FIG. 2.1 Interface utilisateur graphique de l'application de développement et de simulation

2.2.2 Interface pour la construction de réseaux de neurones

L'interface utilisateur graphique permet la construction de réseaux de neurones arbitraires. Une partie de l'interface est dédiée à cette fin. Un menu intercalaire permet l'ajout et la suppression de senseurs, synapses, neurones, regroupements LGPT, couches et actuateurs. De plus, l'interface permet le choix et la modification de leurs paramètres. On présente maintenant toutes ces possibilités en procédant des entrées (senseurs) aux sorties (actuateurs).

2.2.2.1 Senseur

Un senseur est une entrée de tension analogique au réseau de neurones. Un senseur peut être soit une tension analogique directement contrôlée par l'usager, soit une tension analogique provenant d'un senseur sur un robot. Concernant ce dernier, plus de détails seront données au chapitre 3.

La case 1 de la figure 2.2 montre les différentes options de configuration pour un senseur. Chaque senseur a un identificateur unique, commençant par 0, ainsi qu'un nom choisi par l'usager. Dans le cas d'un senseur *Manual*, c'est à dire controlé par l'usager, il n'y a aucun autre paramètre. Les autres types de senseurs sont décrites à la section 3.1.3.

2.2.2.2 Synapse

Une synapse représente justement le circuit d'une synapse. Puisqu'il y a deux modèles synaptiques, il y a deux choix de synapse : *Sensory Nerve* ou *Pulsed Synapse*. Le premier est une synapse qui prend des tensions analogiques comme entrée, et le deuxième est une synapse qui prend des impulsions comme entrée.

La case 2 de la figure 2.2 montre les différentes options de configuration pour une synapse avec entrée analogique (*SensoryNerve*). Son paramètre principal est son poids. Il faut évidemment spécifier le senseur auquel la synapse est branchée et le neurone auquel il appartient. Un neurone est spécifié par la couche (L), le regroupement LGPT (M) et l'identificateur de neurone (N).

La case 3 de la figure 2.2 montre les différentes options de configuration pour une synapse qui prend des impulsions comme entrée (ainsi nommée simplement *Synapse*). Encore ici, son paramètre principal est son poids. Il faut spécifier le

neurone auquel la synapse est branchée ainsi que le neurone auquel il appartient, toujours par leur couche (L), leur regroupement LGPT (M) et leur identificateur (N).

2.2.2.3 Neurone

Le corps du neurone est le circuit somatique. La case 4 de la figure 2.2 montre ses différentes options de configuration. Il faut spécifier sa couche (L) et son regroupement LGPT (M). Chaque neurone a un identificateur unique dans son regroupement LGPT, commençant par 0, ainsi qu'un nom choisi par l'usager. Il y a deux autres paramètres pour un neurone : le seuil de charge pour émettre une impulsion (23nC par défaut) et le courant de fuite (zéro par défaut).

2.2.2.4 Regroupement le-gagnant-prend-tout (LGPT)

Les options de configuration pour un regroupement LGPT sont présentées à la case 5 de la figure 2.2. Il ne faut que spécifier la couche du regroupement. Chaque regroupement LGPT a un identificateur unique dans sa couche, commençant par 0.

2.2.2.5 Couche

Une couche est simplement un regroupement unidimensionnel de neurones. Le seul paramètre d'une couche est le nom choisi par l'usager (voir figure 2.2, case 6).

2.2.2.6 Actuateur

Un actuateur représente une partie contrôlable d'un robot. Les actuateurs sont présentés à la section 3.1.4. La case 7 de la figure 2.2 montre les différentes options de configuration d'un actuateur. Chaque actuateur a un identificateur unique, commençant par 0, ainsi qu'un nom choisi par l'usager. En plus, l'actuateur est branché à la sortie d'un neurone spécifié par la couche (L), le regroupement LGPT (M) et l'identificateur de neurone (N).

2.2.3 Interface de contrôle de simulation

L'interface graphique permet de contrôler la simulation. En fait, elle comprend deux interfaces : une pour contrôler le déroulement et la vitesse de la simulation, et l'autre pour manipuler les entrées du réseau de neurones. La figure 2.3 montre ces deux interfaces (*Control* et *SensoryControl* respectivement). Dans la première, il est possible de faire rouler la simulation avec vitesse ajustable et de l'arrêter, ou de la faire dérouler itération par itération. Dans la deuxième, il est possible d'ajuster la tension de chacune des entrées *SensoryNerve* en manipulant un glisseur.

2.2.4 Interface graphique du comportement du réseau de neurones

L'interface graphique affiche le réseau de neurones dans sa totalité, avec ses paramètres et son comportement en format graphique. Elle est présenté à la figure 2.4 pour un réseau de neurones à deux couches. L'interface montre les senseurs (S), les couches (L), les regroupements le-gagnant-prend-tout (M) et les actuateurs (A) comme des cases avec titres. Les neurones (N) sont dessinés pour être semblables à des neurones biologiques. Finalement, les synapses sont représentées par des lignes

noires qui branchent la source avec le neurone. Les poids synpatiques (en mV) sont écrits à la gauche des neurones.

Pour illustrer le comportement du réseau, plusieurs variables sont affichées. Les tensions analogiques des senseurs sont affichées dans les cases des senseurs. Plus la tension est élévé, plus le chiffre est rouge, et plus elle est faible, plus le chiffre est vert. La quantité de charge dans chaque soma est représentée par un thermomètre situé au centre de chaque neurone. Les sorties neuronnales sont affichées comme un historique des 50 dernières itérations à droite de chaque neurone. Les impulsions sont indiquées comme des lignes verticales.

2.2.5 Interface d'enregistrement et de chargement

L'interface graphique permet à l'usager d'enregistrer les spécifications d'un réseau de neurones particulier et de charger ultérieurement ces enregistrements dans l'application. La case 8 de la figure 2.2 montre l'interface d'enregistrement et de chargement.

2.3 Conclusion

Dans ce chapitre on a présenté la plateforme de développement et simulation des circuits de neurones à impulsions du chapitre 1. En se servant de ce logiciel, il devient possible de rapidement concevoir des réseaux de neurones arbitraires et de simuler leur comportement à haut-niveau. On a donc les outils en main pour pouvoir simuler et tester ces derniers sur des applications embarquées. Au chapitre suivant, on présente le robot qu'on souhaite contrôler par ces circuits de neurones à impulsions et la tâche sur laquelle on mesurera sa performance.

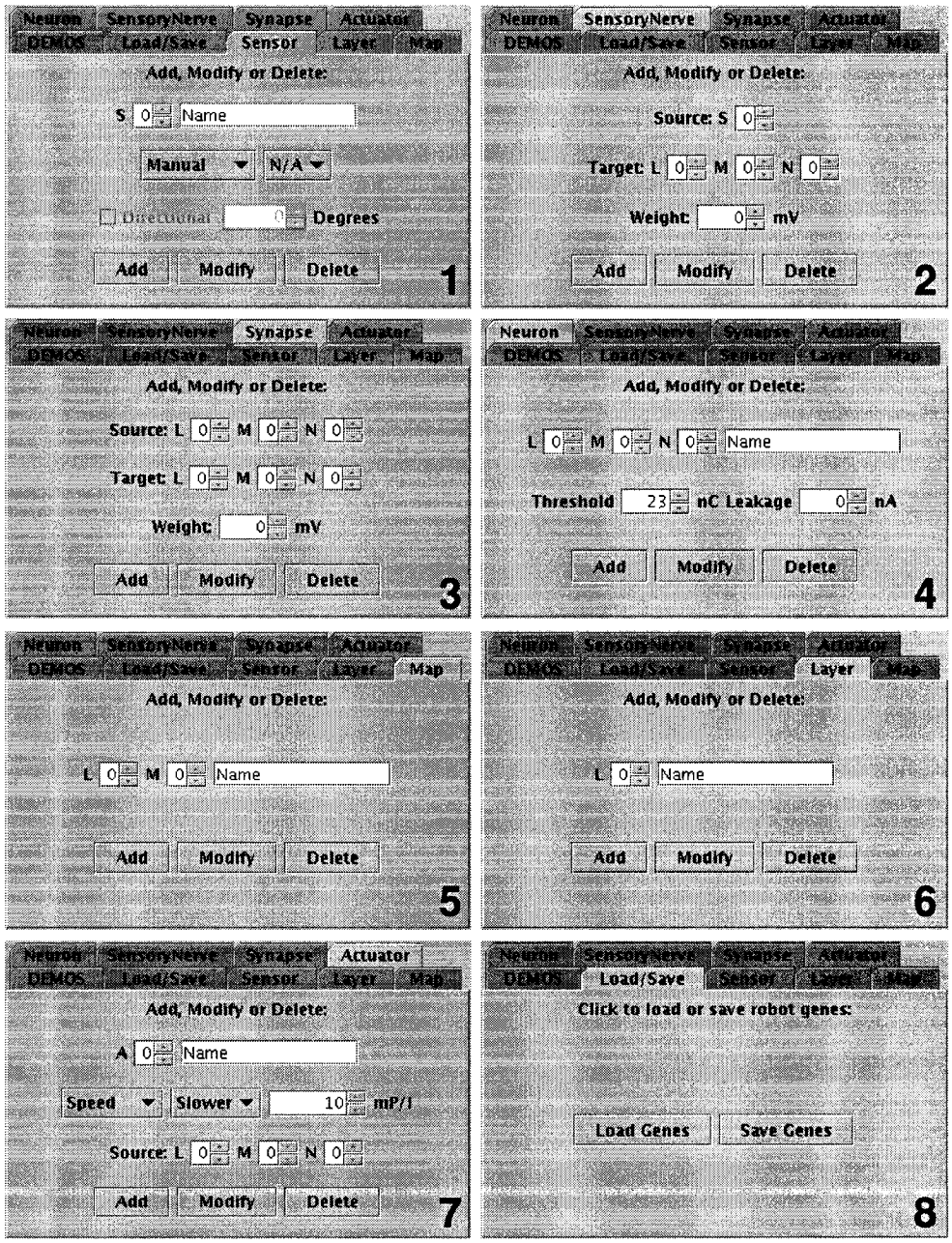

FIG. 2.2 Menu intercalaire pour la construction du réseau de neurones

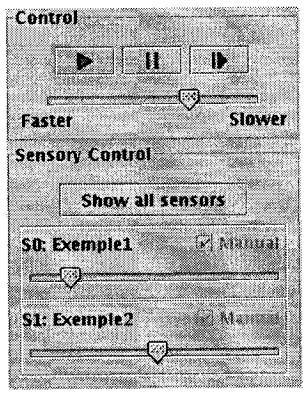

FIG. 2.3 Interface de contrôle

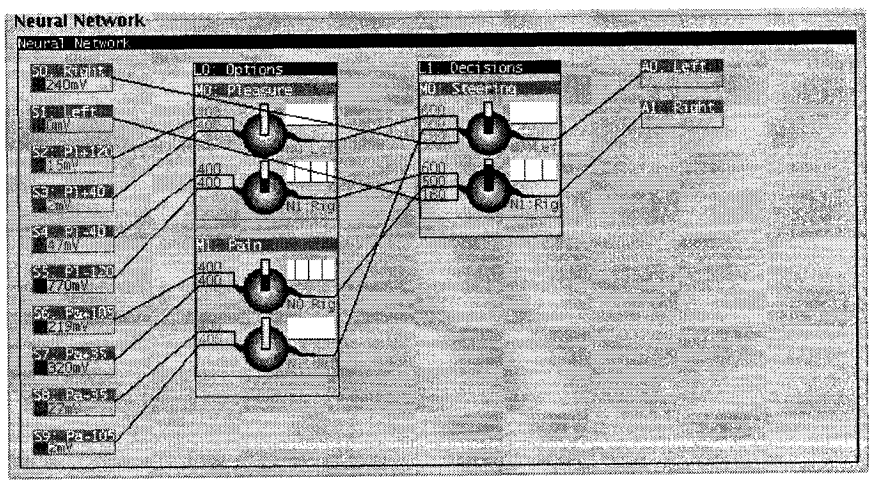

FIG. 2.4 Interface graphique du comportement du réseau de neurones

CHAPITRE 3

APPLICATION EMBARQUÉE : UN ROBOT ET SON ENVIRONNEMENT

Dans ce chapitre on présente l'application embarquée utilisée pour le développement et le test des réseaux de neurones à impulsions simulés. Étant donné que les circuits neuronnaux présentées au chapitre 1 sont inspirées de la biologie, il serait intéressant que le problème sur lequel ils sont appliqués soit également inspiré de la biologie. Nous avons choisi de simuler un robot qui chasse une cible mobile dans un environnement limité. Pour les raisons discutés à la section 3.1.1, ce robot a peu de choses en commun avec les animaux, mais la tâche de chasser une cible mobile est universelle dans le royaume animal. De plus, le réseau de neurones, autrement dit le *cerveau* du robot, est responsable à 100% du comportement du robot. C'est à dire que ce sera le seul système entre ses senseurs et ses actuateurs.

Nous avons également évalué la performance de nos réseaux de neurones à impulsions sur un problème standard en apprentissage machine, soit le OU-exlusif, ainsi que sur la poursuite et fuite simultanée, que nous présenterons dans les sections 5.1 et 5.2.1, respectivement.

3.1 Robot simulé

Il existe actuellement plusieurs types de robots utilisés pour la recherche. Les robots modélisés sur les animaux incluent des humanoïdes, des hexapodes, le celèbre chien Aibo de Sony, etc. Ceux-ci donnent plus l'impression d'êtres vivants, mais ils sont

typiquement difficiles à contrôler. Les robots modélisés sur des véhicules standards incluent des robots à deux, quatre et six roues, des véhicules à chenilles, les robots Khepera, les véhicules de Braitenberg (Braitenberg, 1993), etc. Ceux-ci sont plus faciles à contrôler, mais ressemblent moins à un animal artificiel. Dans les sections suivantes, on présente le choix du robot comme application embarquée adaptée à notre type de réseau de neurones, suivi d'une description de sa dynamique, de ses senseurs et de ses actuateurs.

3.1.1 Choix du robot et sa méthode de propulsion

Étant donné que le *cerveau* du robot est inspiré de la biologie, il serait intéressant que le robot soit inspiré du royaume animal. Une des premières expériences de ma maîtrise était d'essayer de concevoir un robot à partir de *Muscle Wires*®. Les *Muscle Wires*®sont des fils contractibles par un courant électrique qui ont des propriétés semblables aux muscles des animaux. Ils ont l'avantage de pouvoir être contrôlés directement par les impulsions électriques émises par les neurones. Il est possible de construire des robots qui marchent en se servant de deux *Muscle Wires*®pour chaque jambe, comme celui présenté à la figure 3.1. Mais, malheureusement, ces fils contractibles se sont montrés difficiles à manipuler en raison de leur fragilité, en plus d'être lents à réagir et peu puissants (Chénier, 2005). On a donc décidé d'utiliser un robot avec une méthode de propulsion plus robuste et moins exotique.

Des méthodes de propulsion plus communes, mais peu inspirées de la biologie, incluent des systèmes hydrauliques, des servomoteurs, des moteurs électriques, etc. Une technologie pouvant être directement controlée par des impulsions émises par les neurones est le moteur à pas. Un moteur à pas tourne d'un nombre fixe de degrés à chaque fois qu'il reçoit une impulsion à son entrée. Ce type de moteur a

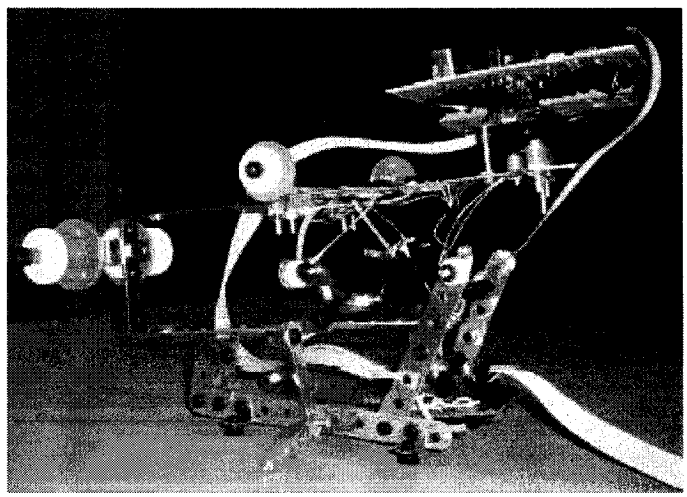

FIG. 3.1 Robot à fils contractibles, construit par Félix Chénier

deux entrées : une qui correspond à chaque direction de rotation.

Pour pouvoir utiliser des moteurs à pas comme méthode de propulsion, il faut trouver une méthode pour diriger cette propulsion dans une direction voulue. Un véhicule existant répondant à ce critère et qui est simple et répandue est la voiture téléguidée. Ce jouet utilise un signal pour contrôler la direction et un autre pour contrôler le papillon des "gaz". Typiquement, ces moteurs sont des servomoteurs, mais ils pourraient également être des moteurs à pas. Puisque la dynamique d'une telle voiture est bien connue et facile à simuler, et puisqu'il serait facile et peu coûteux de construire et d'implanter ce type de robot, nous avons décidé de l'utiliser. La figure 3.2 montre une voiture téléguidée typique avec ses systèmes de propulsion et de direction.

3.1.2 Objectif du robot

Étant donné que le robot visé est une voiture avec direction et vitesse variable, il faut alors lui trouver un objectif qui exploite ses qualités, et, si possible, qui

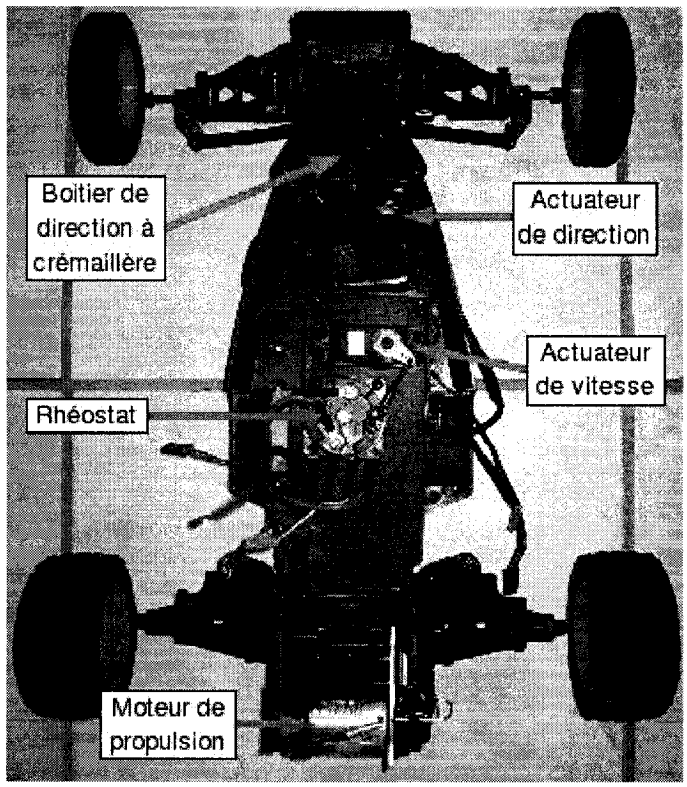

FIG. 3.2 Véhicule sur lequel le robot est basé

soit inspiré de la biologie. Un comportement qui se retrouve universellement dans la nature est la poursuite d'une cible mobile : les prédateurs suivent leur proie, des bébés suivent leur mère, les animaux en général suivent le troupeau, et les mâles ont souvent à poursuivre une femelle pour se reproduire (voir la figure 3.3). Puisque le robot visé a suffisamment de mobilité pour naviguer facilement dans son environnement, ceci représente une tâche bien adaptée à ses capacités.

3.1.3 Senseurs

Avant que le robot ne puisse suivre une cible mobile, il a besoin de la détecter. Dans cette section, on présente les senseurs reliés au robot pour qu'il puisse trouver la cible et naviguer dans son environnement. Comme pour le choix du robot, on simule

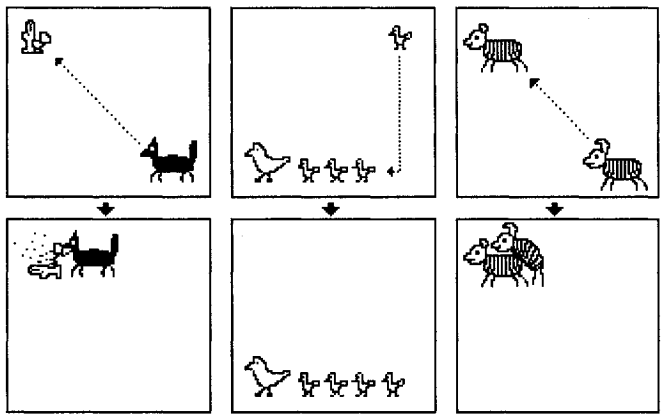

FIG. 3.3 Quelques exemples de la poursuite dans le royaume animal

des senseurs qui sont réalistes pour permettre une implantation future en matériel. Les modèles simulés ne sont pas basés directement sur des senseurs spécifiques, mais plutôt sur leur comportement général. Pour simplifier la simulation, les modèles des senseurs ne sont pas bruités.

3.1.3.1 Senseurs de détection de la cible

Pour que le robot puisse détecter et localiser une cible mobile, on lui raccorde quatres senseurs directionnels et un senseur de proximité. La dynamique de chaque type de senseur est présentée à la figure 3.4[1]. L'orientation des senseurs directionnels sur le robot est présentée à la figure 3.5. Le nombre et l'orientation des senseurs directionnels ont été choisis par essai-erreur : quatres senseurs directionnels orientés vers les quatres coins est le minimum nécessaire pour éviter les angles morts. Dans l'application de simulation, il est possible de spécifier l'orientation de chaque senseur directionnel par rapport à l'axe de la voiture, tel que présentée à la figure 3.6[2]. Les senseurs sont orientés plus vers l'avant du robot pour donner une meilleur résolution d'orientation de la cible (qui normalement devrait être en face

[1]La distance maximale entre deux points dans l'environnement est 340 pixels
[2]Dans l'application, *Pleasure* correspond à la cible mobile

du robot).

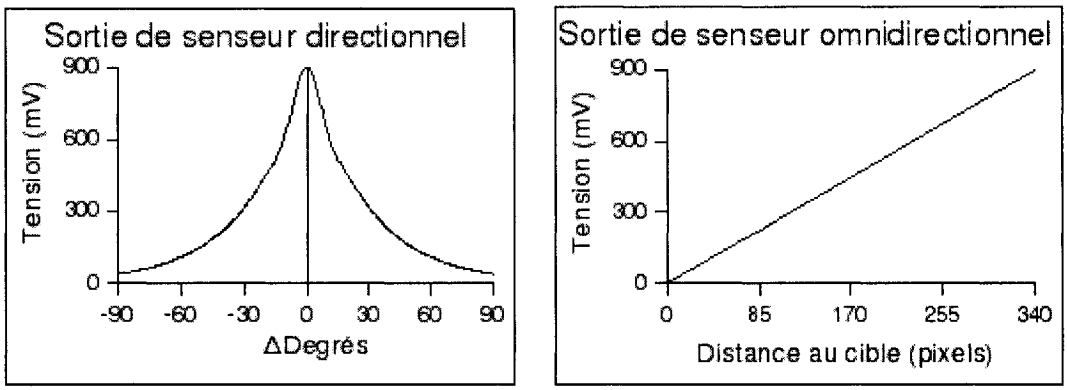

FIG. 3.4 Dynamique des senseurs de détection de la cible

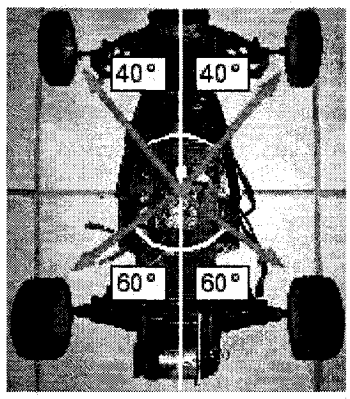

FIG. 3.5 Orientation prédéterminée des senseurs directionnels sur le robot

3.1.3.2 Senseurs de rétroaction

Pour que le robot puisse se conduire cybernétiquement dans son environnement, on raccorde des senseurs de rétroaction sur sa direction et sa vitesse. Sans ces rétroactions, le contrôleur serait en boucle ouverte, et par conséquent, peu performant. Deux senseurs sont utilisés pour indiquer l'état de la crémaillère. Deux senseurs sont utilisés pour indiquer la vitesse de la voiture. La dynamique de chaque type de senseur est présentée à la figure 3.7.

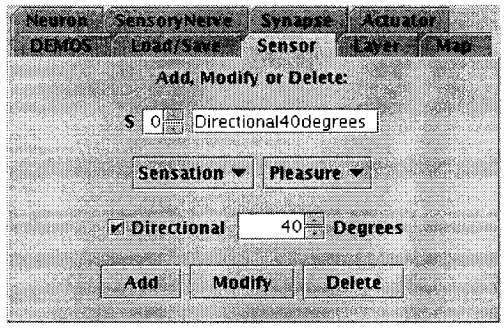

FIG. 3.6 Application de simulation : options pour un senseur directionnel

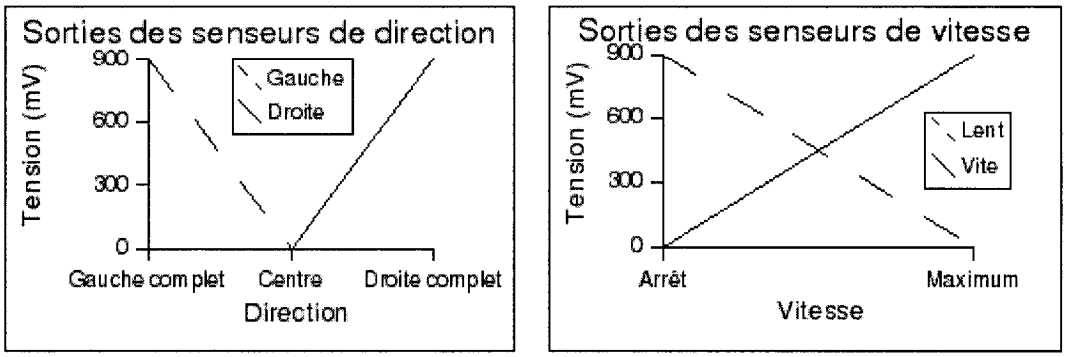

FIG. 3.7 Dynamique des senseurs de détection de la cible

3.1.4 Actuateurs

Le robot comprend deux systèmes qui gèrent son comportement. Sa direction est affectée par un boîtier de direction à crémaillère et sa vitesse est affectée par un rhéostat branché à un moteur électrique de propulsion. Des moteurs à pas sont utilisés comme actuateurs dans les deux cas. Ces moteurs contrôlent la position de l'engrenage et du rhéostat, respectivement. Quand l'actuateur de direction reçoit une impulsion, il change la direction des roues avants par un nombre de degrés constant. Quand l'actuateur de vitesse reçoit une impulsion, il change la vitesse d'un incrément ou décrément constant. La table 3.1 donne les bornes pour ces deux paramètres, ainsi que les valeurs utilisées actuellement pour le robot présenté. La table inclue aussi les bornes sur la vitesse et le ΔCap du robot (à vitesse maximale). Dans l'application de simulation, il est possible de choisir l'amplitude de l'effet des

TAB. 3.1 Effet des actuateurs sur la direction et la vitesse

	Minimum	Maximum	Robot Actuel
ΔVitesse (mPixels/Iteration)	10	1000	30
ΔDirection (dDegrés/Iteration)	1	10	9
Vitesse (mPixels/Iteration)	0	1200	n/a
ΔCap (dDegres/Iteration)	-90	90	n/a

actuateurs, comme présenté à la figure 3.8, dans les bornes de la table 3.1. En réalité, ceci correspond à changer l'engrenage de direction et le ratio de couplage du moteur à pas avec le rhéostat.

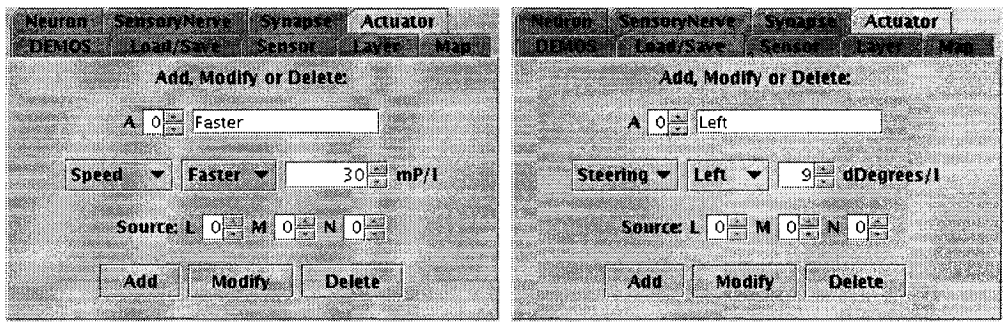

FIG. 3.8 Application de simulation : options pour les actuateurs

Le cap du robot est calculé à chaque itération (i) selon l'équation 3.1 :

$$Cap_i = Cap_{i-1} + \Delta\text{Direction}_{i-1} \times \frac{Vitesse_{i-1}}{Vitesse_{maximale}} \tag{3.1}$$

3.2 Environnement simulé

L'environnement simulé représente le système dans lequel le robot et la cible mobile co-existent. La figure 3.9 montre l'environnement, le robot, la cible mobile et une troisième véhicule. Le robot est au centre et la cible mobile est dans le coin en bas à gauche. Le troisième véhicule en haut à droite sert comme reférence et n'a

aucun effet dans la poursuite autre que déterminer la distance moyenne entre deux mobiles qui se promènent aléatoirement, comme décrit à la section 3.2.4 (un rôle alternatif de ce mobile est présenté à la section 5.2.1). Chaque simulation débute avec les véhicules positionnés et orientés de cette façon.

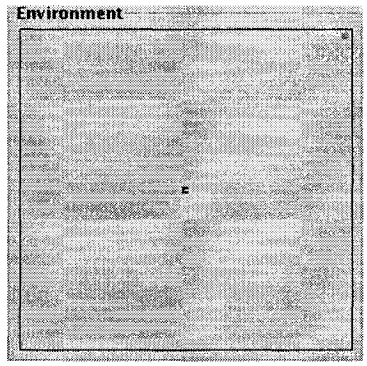

FIG. 3.9 Application de simulation : affichage de l'environnement

3.2.1 Dynamique de l'environnement

Le robot et la cible mobile peuvent se promener n'importe où à l'intérieur du rectangle, mais ils ne peuvent pas franchir ses murs. Quand l'un ou l'autre des véhicules rejoint un mur, il rebondit avec la même vitesse sur un cap aléatoire dirigé vers l'intérieur du rectangle. Ceci permet de vérifier que le robot peut retrouver la cible même s'il y a un discontinuité dans la poursuite, et donc dans la valeur de ses senseurs. Il n'y a aucun obstacle dans l'environnement à part les murs. Si le robot et la cible occupent le même espace, il n'y a aucune conséquence. Tout ceci pour assurer que la poursuite de la cible mobile demeure l'unique objectif du robot.

3.2.2 Cible mobile

La cible mobile n'est pas le même véhicule que le robot et donc n'a pas la même dynamique. La cible mobile change aléatoirement sa vitesse et son cap à chaque

TAB. 3.2 Paramètres de la cible mobile

	Minimum	Maximum
ΔVitesse (mPixels/Iteration)	-5	+5
ΔDirection (dDegres/Iteration)	-90	+90
Vitesse (mPixels/Iteration)	100	400

itération. Les valeurs maximales de ces changements, ainsi que les bornes sur la vitesse de la cible mobile sont inscrites à la table 3.2. Les valeurs aléatoires de ΔVitesse et ΔDirection sont distribuées également entre les valeurs minimales et maximales présentées. Par conséquent, la cible se déplace érratiquement, ce qui la rend difficile à suivre. Le véhicule de référence de l'environnement possède la même dynamique que la cible mobile.

3.2.3 Dynamique de la poursuite

En comparant la dynamique du robot et celle de la cible mobile, on note que le robot a une avantage en vitesse et en accéleration et décéleration. Le robot est trois fois plus vite et il peut ajuster sa vitesse six fois plus rapidement. Par contre, la cible mobile a un avantage en agilité. La cible mobile peut effectuer des virages dix fois plus serrés que le robot. Résultat : le robot est capable de ratrapper la cible en ligne direct mais la cible peut tourner plus vite pour lui échapper. En la nature on retrouve des relations semblables : lynx contre lapin, guépard contre springbok, etc. Ces ratios de performance, choisis par essai-erreur, assurent que la poursuite demeure toujours un défi pour le robot, ce qui permet de tester constamment son *cerveau* de circuits de neurones à impulsions de type *leaky integrate-and-fire*.

3.2.4 Mesure de performance

L'objectif du robot est la poursuite constante de la cible mobile. La mesure de performance correspondante est donc la distance moyenne entre le robot et la cible mobile. Plus le robot s'approche de la cible, plus on dira que son *cerveau* est performant. Cette distance moyenne est calculée et mise à jour à chaque itération de la simulation. La valeur est affichée dans l'application, tel que présentée à la figure 3.10. Évidemment, plus la simulation se déroule, moins cette valeur varie.

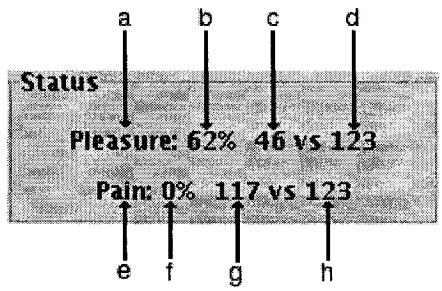

FIG. 3.10 Application de simulation : mésure de performance

Pour l'application embarquée décrite dans ce chapitre, seule la première ligne (a - d) de la mesure de performance nous intéresse. La deuxième ligne (e - h) correspond plutôt au cas spécial où le véhicule de référence est un mobile *à éviter*, comme un prédateur du robot. Pour cette raison, le véhicule de référence est nommé *Pain* (la douleur) dans l'application de développement et de simulation. On discute de cette configuration à la section 5.2.1. Dans notre configuration, le véhicule de référence sert à déterminer la distance moyenne entre deux véhicules qui se promènent aléatoirement dans l'environnement. Ces mesures nous permettent d'estimer la performance du contrôleur du robot par rapport à la chance. L'équation 3.2 indique comment la côte de plaisir est calculée :

$$\text{Cote de plaisir} = \frac{DistanceMoy_{reference-cible} - DistanceMoy_{robot-cible}}{DistanceMoy_{reference-cible}} \times 100\% \tag{3.2}$$

Description des paramètres de la figure 3.10 :

a) **Cible mobile** : la ligne intitulée *Pleasure* (le plaisir) correspond à la cible mobile

b) Côte de plaisir : le pourcentage qui représente comment c est mieux que d

c) Distance moyenne entre le robot et la cible mobile

d) Distance moyenne entre le véhicule de référence et la cible mobile

e) **Véhicule de référence ou prédateur** : la ligne intitulée *Pain* (le mal) correspond au véhicule de référence, ou alternativement comme prédateur du robot (voir section 5.2.1)

f) Côte de douleur : le pourcentage qui représente à quel point g est mieux que h

g) Distance moyenne entre le robot et le véhicule de référence

h) Distance moyenne entre la cible mobile et le véhicule de référence

3.3 Conclusion

Dans ce chapitre on a présenté un simple robot simulé qu'on espère contrôler par les neurones à impulsions présentés au chapitre 1. En plus, on a présenté comment il est possible de simuler ce robot et son environnement avec la plateforme de développement et de simulation présentée au chapitre 2. Il reste à utiliser cette plateforme pour concevoir un réseau de neurones qui permet au robot de suivre la cible mobile de très près, ce qui est le sujet du chapitre suivant.

CHAPITRE 4

DÉVELOPPEMENT ET OPTIMISATION DU ROBOT ET DE SON RÉSEAU DE NEURONES

On présente maintenant la méthode utilisée pour développer un contrôleur de robot (c'est à dire un réseau de neurones) performant. Les premiers contrôleurs sont conçus expérimentalement dans l'application de développement et de simulation. Ceux-ci donnent des résultats acceptables, mais pour obtenir un robot performant, il faudrait appliquer un algorithme d'optimisation automatique, capable d'explorer l'immense espace de solutions de façon opportuniste. On développe alors une application à interface usager graphique qui implante un algorithme évolutif (d'optimisation). Avec cet outil, on optimise les premiers contrôleurs jusqu'à ce que le robot soit capable de suivre la cible de plus près possible.

4.1 Réseau de neurones initial

Les premiers contrôleurs de robot ont été conçus expérimentalement dans l'application de développement et de simulation, décrite au chapitre 2. Dans cette section, on présente le contrôleur le plus performant parmi ceux-ci, et les étapes ayant mené à sa conception.

4.1.1 Choix d'architecture

La première question dans la conception d'un réseau de neurones concerne typiquement son architecture. Combien de neurones en faut-il ? Combien de couches ?

Quelles interconnexions ? Ce sont les questions qu'il fallait répondre en développant le premier contrôleur. Notre approche fut de commencer avec l'architecture la plus simple, puis de la raffiner si nécessaire. Le robot a quatre actuateurs (droite, gauche, accélère et décélère) qui nécessitent chacun un neurone pour lui envoyer des impulsions. L'architecture la plus simple comprend donc une seule couche de quatre neurones branchés aux actuateurs.

Étant donnée cette configuration de neurones, il fallait spécifier leurs entrées. Afin de commencer avec une architecture minimale, on a décidé de ne pas brancher chaque neurone à chaque senseur. On a plutôt pris l'approche d'ajouter des connexions synaptiques au fur et à mesure qu'elles amélioraient la performance du réseau. Le choix des connexions a été effectué en suivant les deux astuces naturelles suivantes :

- lier les senseurs de la cible aux neurones qui dirigeront le robot vers la cible (exemple : la cible est à gauche donc tourne plus vers la gauche)
- lier les senseurs de vitesse et de direction aux neurones responsables de l'action inverse (exemple : on roule vite donc ralentissez, on tourne fortement à gauche donc tourne vers la droite)

La première astuce nous assure que le robot s'oriente vers la cible mobile. La deuxième astuce nous assure que le contrôleur soit stable.

En effet, la rétroaction négative, comme prescrit par la deuxième astuce, est souvent utilisé en automatique pour assurer qu'un système soit stable et non-oscillatoire. Quant au robot, en l'absence d'une cible mobile, seul ses senseurs de rétroaction seront actifs. Avec de la rétroaction négative, le robot se stabiliserait vers une direction et vitesse fixe parce que ses senseurs de direction et de vitesse ont chacun un effet inverse sur les actuateurs. Sans cette rétroaction, le robot ne pourrait jamais

changer de direction ni de vitesse, et il se déplacerait probablement en cercles. Le truc est donc d'appliquer les deux astuces précédentes de façon judicieuse : il faut trouver un bon compromis entre temps de réaction et stabilité (et évidemment éviter des comportements oscillatoires). La figure 4.1 présente, de manière générale, l'effet de la rétroaction négative sur un déplacement vers une cible fixe. Pour un bon texte d'introduction à la théorie du contrôle, voir (Dorf et Bishop, 2000).

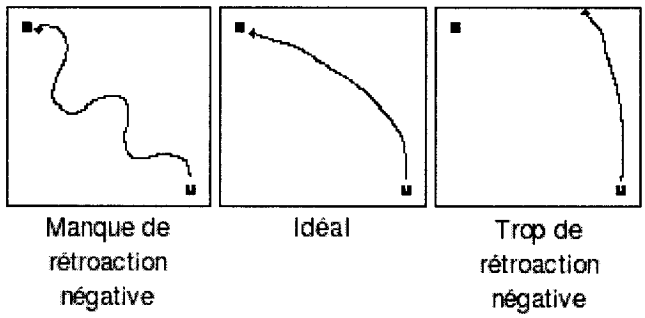

FIG. 4.1 Effets de la rétroaction négative sur un déplacement vers une cible fixe

La table 4.1 présente le comportement souhaitable du robot selon plusieurs scénarios. Les liens correspondants entre senseurs et actuateurs qui améliorent la performance sont présentés à la figure 4.2. Les senseurs sont listés dans chaque case correspondante à un des quatres neurones d'actuateur (accélère, tourne à droite, décelère, tourne à gauche).

4.1.2 Ajustement des paramètres

L'architecture du réseau étant choisie, il restait à préciser ses paramètres tels que les poids synaptiques, la fuite des neurones, et le seuil de charge. Étant donné que la performance du robot est très sensible aux valeurs des poids synaptiques, nous avons concenté notre étude sur la détermination de ceux-ci, en fixant les autres paramètres. Les poids synaptiques ont été choisis, par essais et erreurs, en tentant de respecter les critères généraux, qualitatifs et souvent opposés suivants, soit de

TAB. 4.1 Comportement souhaitable selon la situation

Scénario	Action
Vitesse lent Direction non-centrée	Accelère
Cible très près Vitesse vite Cible en arrière	Décelère
Cible à droite Direction à gauche Vitesse lent Cible très près	Tourne à droite
Cible à gauche Direction à droite Vitesse lent Cible très près	Tourne à gauche

définir les poids synaptiques du (des) senseur(s) :

- de la cible pour que le robot tourne vers la cible. (En gardant la vitesse du robot fixe pour cette étape et la prochaine.)
- de direction sur les neurones de direction (inverse) pour que les virages soient réalisés sans oscillation.
- de proximité sur le neurone de décélération pour éviter que le robot ne dépasse la cible quand il suit la cible de très près.
- de vitesse sur les neurones de vitesse (inverse) pour que le robot atteigne une vitesse "appropriée".
- de lenteur sur les neurones de direction pour aider le robot à changer de cap rapidement quand il roule lentement[1].
- de direction sur le neurone d'accélération pour aider le robot à changer rapidement de cap.

[1]Même si la direction est complètement à droite ou à gauche, il faut rouler pour changer de cap.

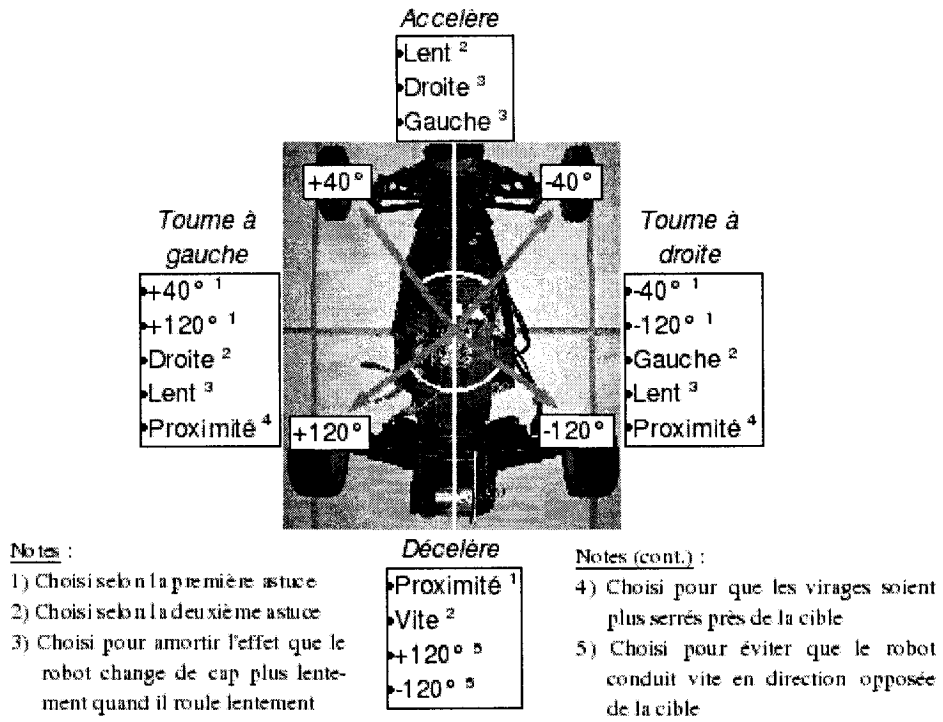

FIG. 4.2 Connexions synaptiques du réseau, choisies par intuition

- de proximité sur les neurones de direction pour que le robot devienne plus agile quand il approche la cible[2].
- de direction orientés vers l'arrière du robot sur le neurone de décélération pour éviter que le robot se distance trop de la cible.

Un réseau de neurones avec des poids synaptiques adéquats est présenté à la figure 4.3 (poids synaptiques en mV, à gauche des neurones). Ceux-ci sont aussi présentés dans la table 4.2.

Ayant précisé des poids synaptiques selon les critères précédents, il restait à ajuster les fuites et les seuils de charge des neurones. Ces deux paramètres ont des effets semblables sur la fréquence d'impulsion d'un neurone pour des entrées données.

[2]Si la cible se déplace perpendiculairement au cap du robot quand les deux sont près, il faut que le robot tourne rapidement pour la garder toujours en face

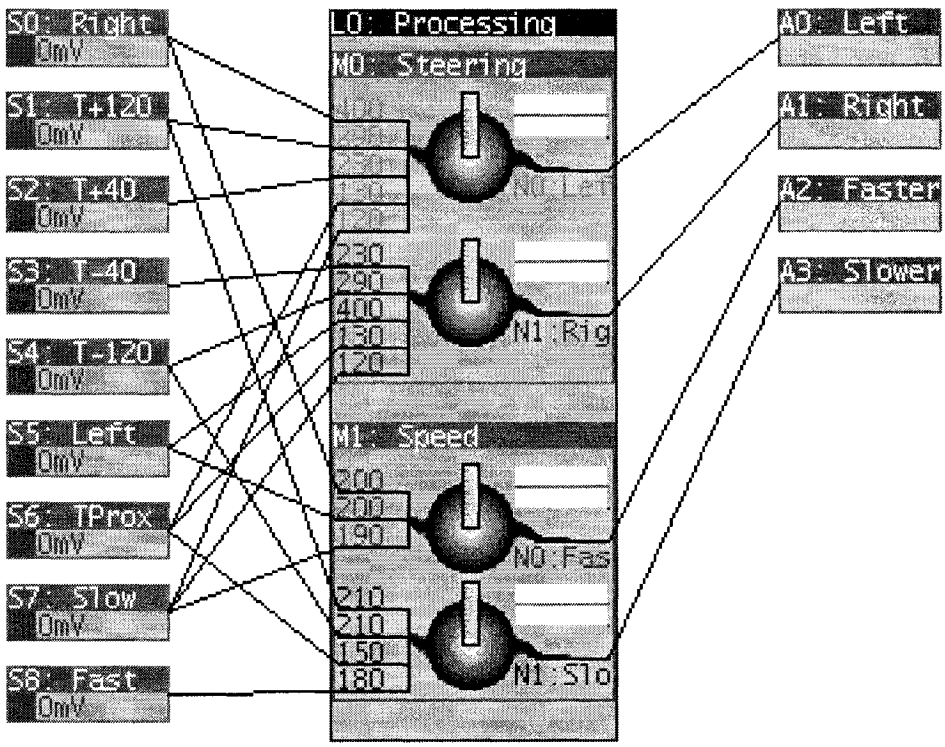

FIG. 4.3 Poids synaptiques du réseau, choisis par approximation qualitative

La fuite a l'effet supplémentaire d'éviter qu'une activité transitoire sur les entrées produise une impulsion indésirée. Pour cette raison, on a décidé de garder les seuils de charge par défaut et de plutôt ajuster les fuites pour améliorer le comportement du robot. Ces fuites, choisies par essais et erreurs, sont présentées dans la table 4.3.

4.1.3 Performance et comportement du contrôleur initial

Il nous a pris environ deux semaines d'expérimentation pour développer la liste des astuces de la section 4.1.2 et de préciser les poids synaptiques et fuites somatiques. Le réseau de neurones résultant permet au robot de suivre une cible mobile à une distance moyenne de 44 pixels[3]. Par contre, un robot qui se promène aléatoirement

[3]Rappel : l'environnement mesure 245 par 235 pixels

TAB. 4.2 Poids synaptiques (en mV) du réseau de neurones initial

	Neurone			
Senseur	Gauche	Droite	Accélère	Décélère
Droite	400	-	200	-
+120°	290	-	-	210
+40°	230	-	-	-
-40°	-	230	-	-
-120°	-	290	-	210
Gauche	-	400	200	-
Prox.	130	130	-	150
Lent	120	120	190	-
Vite	-	-	-	180

TAB. 4.3 Fuite et seuil de charge des neurones du contrôleur initial

Neurone	Fuite	Seuil de charge
Direction - Gauche	10nA	23nC
Direction - Droite	10nA	23nC
Accélération	20nA	23nC
Décélération	20nA	23nC

dans l'environnement n'atteint qu'une distance moyenne de 127 pixels. Il ne serait pas difficile pour un "étranger" de développer en quelques heures un réseau capable de battre cette distance. Par contre, il est beaucoup moins évident comment obtenir ou battre une distance moyenne de 44 pixels.

Le robot avec le réseau de neurones specifié suit très bien la cible quand elle ne change pas rapidement de vitesse ni de direction. Par contre, lorsque la cible effectue des virages serrés à proximité, le robot accélère souvent trop fort en changeant de direction et dépasse la cible (voir la figure 4.4). Ou encore, lorsque le robot perd la cible, il réagit vite pour l'avoir en face de lui, mais n'accélère pas assez vite pour la rattrapper rapidement. Il y a donc place à l'amélioration, mais étant donné l'effort nécessaire pour le faire par expérimentation, il était souhaitable de développer une plateforme d'optimisation automatique.

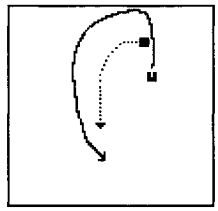

FIG. 4.4 Comportement inattendu du robot initial (solide) en forte proximité de la cible (pointillé)

4.2 Plateforme d'optimisation de robots

4.2.1 Analyse du problème

Le but du robot est de poursuivre une cible mobile de plus près possible, donc la mesure de performance est la distance moyenne entre les deux. La problématique est donc d'optimiser les paramètres du robot et de son réseau de neurones pour minimiser cette distance. Est-il utile de connaître le comportement idéal du robot

à chaque instant dans chaque situation ? La réponse est définitivement non. Étant donné que le comportement de la cible est aléatoire, il est impossible de savoir quel comportement du robot serait idéal, sans savoir les actions futurs de la cible mobile. Même un comportement qui amenera le robot vers la position courante de la cible n'est pas nécéssairement optimal. De plus, la dynamique du robot, et ses neurones, est beaucoup trop complèxe pour quantifier et implanter un tel comportement.

4.2.2 Techniques d'optimisation pour neurones à impulsions

Comme discuté dans l'introduction, nous voulons optimiser le réseau de neurones par évolution opportuniste. Les animaux simples s'adaptent par hasard et sélection aux changements dans leur environnement de génération à génération par évolution. Cette même technique peut facilement s'appliquer à notre robot. D'autres techniques d'optimisation pour neurones à impulsions existent, mais ceux-ci sont le sujet de la recherche actuelle et dépasssent largement le cadre de cette maîtrise (Gerstner et Kistler, 2002). Ces techniques incluent la correction d'erreur par descente de son gradient[4], l'adaptation par la loi de Hebb, etc.

Quant à l'évolution biologique, plusieurs variations existent dans la nature. Il y a des animaux qui se reproduisent sexuellement et d'autres qui le font asexuellement. La génération suivante représente une copie du parent ou de la mélange des parents, avec des erreurs souvent présentes. Des mutations peuvent aussi se produire pendant la vie d'un individu, à cause de la radiation par exemple. Comme pour les animaux les plus simples, on adapte notre simple robot par la reproduction asexuée. L'algorithme évolutif qui implante cette technique est présenté dans la section 4.2.3. Étant donné que notre objectif est simplement d'optimiser le robot

[4]Puisque la fonction de transfert du robot est inconnue, cette technique ne pourrait même pas être appliquée

pour compléter une preuve de concept plutôt que de rechercher les méthodes d'optimisation, on se concentre sur l'implantation d'un algorithme de base. Pour l'état de l'art des techniques génétiques et évolutives, voir (Koza, 2003) et les versions précédentes.

4.2.3 Algorithme évolutif

L'algorithme évolutif implanté est essentiellement une reproduction asexuée avec erreurs stochastiques qui fonctionne ainsi. Un réseau de neurones (le *contrôleur parent* ou CP) est reproduit, avec des variations sur les valeurs de ses paramètres (mutations), pour produire un *contrôleur enfant* (CE). Plusieurs CE du même parent compétitionnent pour chasser la même cible mobile pendant un certain nombre d'itérations. Le CE ayant la plus faible distance moyenne à la cible mobile est le gagnant. Ensuite, la méthode se repète en utilisant le réseau de neurones du CE gagnant comme CP.

Les paramètres qui varient dans la reproduction sont les suivants :
- poids synaptique
- fuite somatique
- seuil somatique
- direction des senseurs
- gain des actuateurs

L'amplitude des variations aléatoires est indépendante pour chaque paramètre. L'équation 4.1 détermine la valeur d'un paramètre d'un CE lors de la reproduction asexuée.

$$Parametre_{enfant} = Parametre_{parent} \times (1 + Erreur_{gaussienne} \times Facteur_{erreur}) \quad (4.1)$$

Dans l'équation, $Erreur_{Gaussienne}$ est un nombre aléatoire qui suit une distribution normale de moyenne zéro et d'écart-type de 1. Le $Facteur_{erreur}$ est variable entre zéro et cent pour chaque paramètre, ce qui peut être choisi par un usager dans l'application évolutive (voir section 4.2.4).

L'algorithme évolutif n'a aucun effet sur l'architecture du réseau, ce qui est biologiquement plausible. Si on désire comparer les performances potentielles de différentes architectures, il est nécessaire d'évoluer chaque architecture indépendemment. Ensuite, les meilleurs contrôleurs de chaque architecture peuvent être comparés.

4.2.4 Interface de l'application évolutive

L'application évolutive possède une interface usager graphique qui facilite l'optimisation d'un réseau de neurones donné, présentée à la figure 4.5. Avec cet outil, l'usager prend un CP d'architecture et performance souhaitable, et le réproduit itérativement jusqu'au point où les CE ne réalisent plus de gains de performance.

L'application évolutive permet à l'usager de charger un fichier qui représente un contrôleur de robot, et de le reproduire par multiplication asexuée. Le nombre de CE à générer, le nombre d'itérations de l'évaluation et les $Facteur_{erreur}$ pour chaque paramètre peuvent êtres choisis à l'intérieur des bornes de la table 4.4. Typiquement, il serait souhaitable d'avoir plus que 12 enfants par génération, mais on se limite à ce nombre pour simplifier leur affichage dans l'environnement[5]

L'application évolutive affiche l'environnement de la même façon que dans l'application de développement et simulation (voir chapitre 2) sauf que le CP et tous les CE y sont affichés. Le CP est noir, et les CE sont d'autres couleurs. La distance

[5]En ne pas affichant les comportements des robots et de la cible, des versions futures de l'application évolutive pourraient supporter beaucoup plus d'enfants par génération.

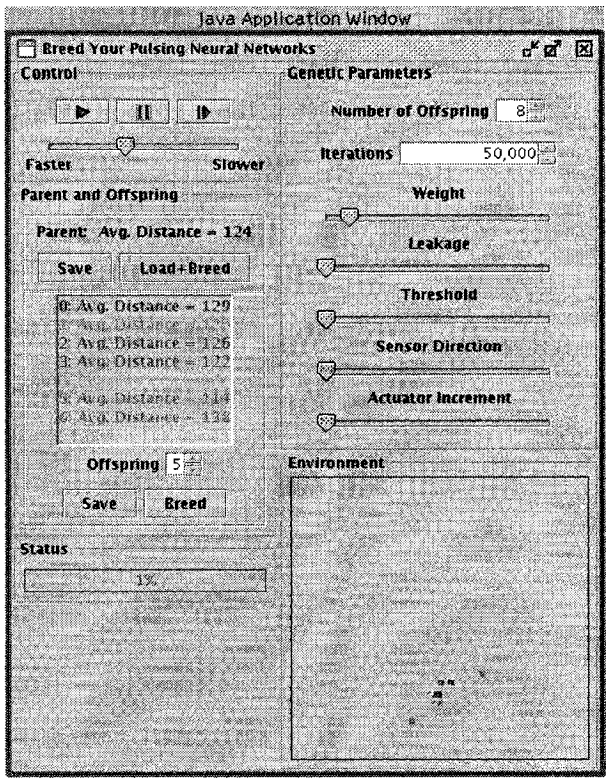

FIG. 4.5 Application évolutive

moyenne entre chaque robot et la cible mobile est affichée et mise à jour à chaque itération. La figure 4.6 montre le panneau *Parent and Offspring* (*Parent et ses enfants*) après le nombre désiré d'itérations. Remarquez que dans le cas illustré, le CP a une distance moyenne de 45 tandis que tous les CE, sauf le 9, ont des distances moyennes égales ou plus élevées. La couleur du texte correspond à la couleur du CE.

TAB. 4.4 Bornes sur les variables de l'algorithme évolutif

	Minimum	Maximum
Enfants	1	12
Itérations	10000	10000000
$Facteur_{erreur}$	0	100

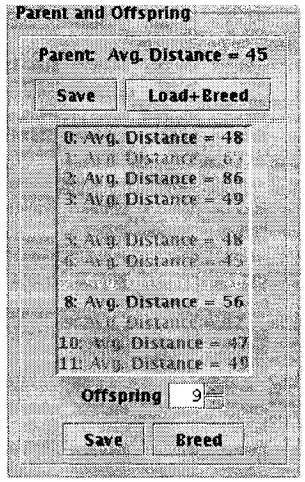

FIG. 4.6 Performance finale du parent et de ses enfants

Pour reproduire un des CE, il suffit de le choisir à l'aide d'un bouton fléché, et d'activer le bouton *Breed* (*Reproduction*). Quand le nombre désiré d'itérations est atteint lors d'une simulation, le CE le plus performant est choisi par défaut comme futur CP, comme c'est le cas pour le CE 9 dans la figure 4.6.

4.2.5 Optimisation des réseaux par évolution

L'algorithme évolutive représente une exploration opportuniste de l'espace des paramètres d'un réseau de neurones d'architecture donnée. L'usager guide l'exploration en précisant l'amplitude des variations. Dans l'évolution avec petites variations, les CE demeurent semblables au CP, ce qui fait que les gains en performance sont typiques, mais généralement faibles. Par contre, dans l'évolution avec grandes variations, il est fortement probable que les enfants souffrent de mutations qui dégradent leur performance. Mais, il y a toujours la très faible chance qu'un des enfants obtienne un gain de performance majeur.

Le dilemme est donc de guider adéquatement l'évolution, ce qui est plus simple que de regler les paramètres par essai-erreur. Si les mutations sont trop sévères, c'est

plus probable d'avoir des pertes en performance. Si les mutations sont trop faibles, on risque de rester pris dans un minimum local. Un bon compromis est donc de maximiser les amplitudes de mutations tout en assurant que typiquement au moins un CE réalise un gain en performance. Alors il est toujours souhaitable d'utiliser le maximum nombre de CE possible. Le nombre d'itérations devrait être assez élevé pour assurer que les distances moyennes se stabilisent à des valeurs fiables, ce qui permet une comparaison de performance juste. Et finalement, le critère de terminaison est défini selon les besoins de l'usager. Par exemple, quand la distance moyenne désirée est atteinte ou quand les distances moyennes n'améliorent plus de génération à génération.

4.3 Optimisation du réseau de neurones

Le contrôleur présenté dans la section 4.1 permet au robot de suivre la cible mobile avec une proximité moyenne beaucoup mieux que par chance (44 versus 127 pixels). Dans cette section, on espère améliorer la performance de ce contrôleur en l'optimisant par l'application évolutive présentée dans la section 4.2.4. Notre approche est de modifier un seul paramètre, les poids synaptiques, avec une magnitude d'erreur qui permet qu'au moins un des CE réalise un gain de performance, et de reproduire le CE qui donne la meilleure performance (même si c'est pire que le CP). Dans cette section, on présente le chemin suivi pour obtenir des contrôleurs plus performants que celui obtenu manuellement.

TAB. 4.5 Paramètres de la première évolution

	Valeur
Nombre de générations	10
Itérations par génération	100 000
Enfants par génération	12
$Facteur_{erreur}$ de poids	10

4.3.1 Optimisation des poids synaptiques sur une évolution de dix générations

On utilise le contrôleur présenté dans la section 4.1 comme CP initial dans l'application évolutive. On le modifie pour que chaque neurone ait une connexion synaptique à chaque senseur pour permettre à de nouvelles synapses d'intervenir au cours de l'évolution. Ces nouvelles connexions synaptiques ajoutées ont un poids par défaut de 50mV pour assurer que leur sortie soit toujours zéro[6]. Dans les générations futurs, ces poids pourraient varier suffisament pour avoir un effet sur le comportement du robot.

Comme premier essai d'optimisation, on fait évoluer ce réseau en utilisant les paramètres donnés dans la table 4.5. Le CE gagnant de chaque génération est utilisé comme CP pour la prochaine. On répète cette procédure au complet trois fois en parallèle pour assurer que les gains de performance sont semblables et répétables. La figure 4.7 montre l'évolution de performance, génération par génération, pour ces trois essais. Les distances moyennes dans cette figure sont les moyennes des deux performances du contrôleur (une fois comme CP et la prochaine fois comme CE)[7].

[6]Des poids synaptiques inférieures à 100mV n'ont aucun effet

[7]Les distances moyennes peuvent différer à cause du comportement aléatoire de la cible

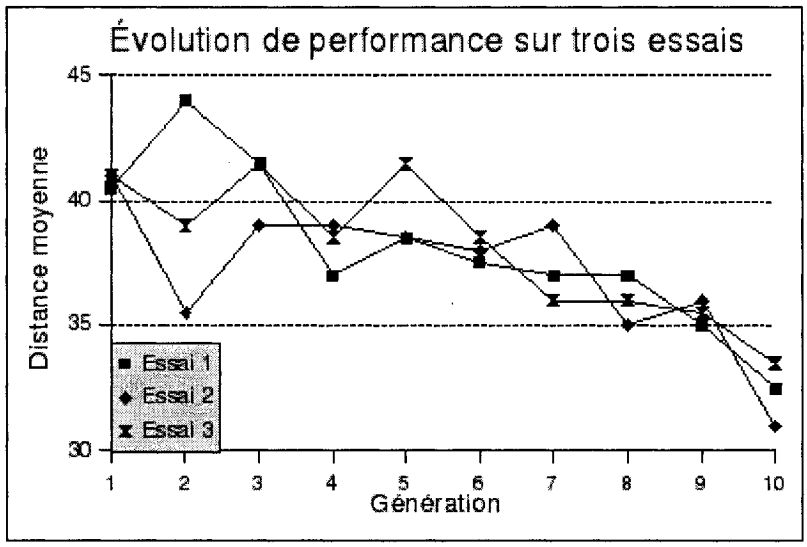

FIG. 4.7 Résultats de la première évolution

TAB. 4.6 Poids synaptiques (en mV) pour les neurones de direction après la première évolution

Neurone gauche				Neurone droite			
Senseur	E. 1	E. 2	E. 3	Senseur	E. 1	E. 2	E. 3
Droite	539	409	355	Gauche	363	405	390
+40°	328	295	311	-40°	362	369	199
+120°	349	337	441	-120°	516	679	322
Prox.	123	196	-	Prox.	-	107	153
Lent	129	-	-	Lent	147	170	-

Les tables 4.6 et 4.7 montrent les poids synaptiques du CE gagnant de la dixième génération de chaque essai, en plus que la moyenne des synapses utiles. Pour les poids qui sont inférieurs à 100mV, on dit que la connexion synaptique est inutile, et on marque ceci par un tiret.

On remarque à la table 4.8 que chaque contrôleur donne un comportement distinct. Ceci nous indique qu'une variété de solutions peuvent mener à des performances semblables. Donc, en plus d'assurer que les résultats sont semblables et répétables, on obtient des robots à comportements variés en faisant trois évolutions parallèles.

TAB. 4.7 Poids synaptiques (en mV) pour les neurones de vitesse après la première évolution

Neurone accélération				Neurone décélération			
Senseur	E. 1	E. 2	E. 3	Senseur	E. 1	E. 2	E. 3
Droite	262	270	111	+120°	169	106	252
Gauche	144	253	187	-120°	138	172	120
Lent	163	168	179	Prox.	123	178	104
n/a	-	-	-	Vite	227	185	193

TAB. 4.8 Description qualitative des comportements des robots après les premiers dix générations

Contrôleur	Description	Dist. Moy.
Essai 1	Change de direction de manière oscillatoire derrière la cible, dépasse souvent la cible, rattrappe vite la cible.	31
Essai 2	Dépasse rarement la cible, accélère fortement dans les virages, suit bien la cible mais à une distance élevée.	32.5
Essai 3	Vitesse très stable (même dans les virages), suit la cible en forte proximité.	33.5

4.3.2 Optimisation des poids synaptiques sur une autre évolution de dix générations

On répète la même procédure que la section précédente (une séquence de dix générations), avec deux exceptions. En premier, on utilise les trois contrôleurs décrits dans la table 4.8 comme CP initiaux. Deuxièmement, on test sur 200 000 itérations au lieu de 100 000 pour mieux assurer que les gains sont dûs aux paramètres et non à la chance. Cette fois-ci on s'attend à des gains plus faibles avec chaque génération puisque les contrôleurs sont déjà raffinés, alors on a besoin de plus d'itérations pour observer des gains.

La figure 4.8 montre l'évolution de performance par génération pour ces trois essais. Les distances moyennes dans cette figure sont la moyenne des deux performances du contrôleur (une fois comme CE et la prochaine fois comme CP).

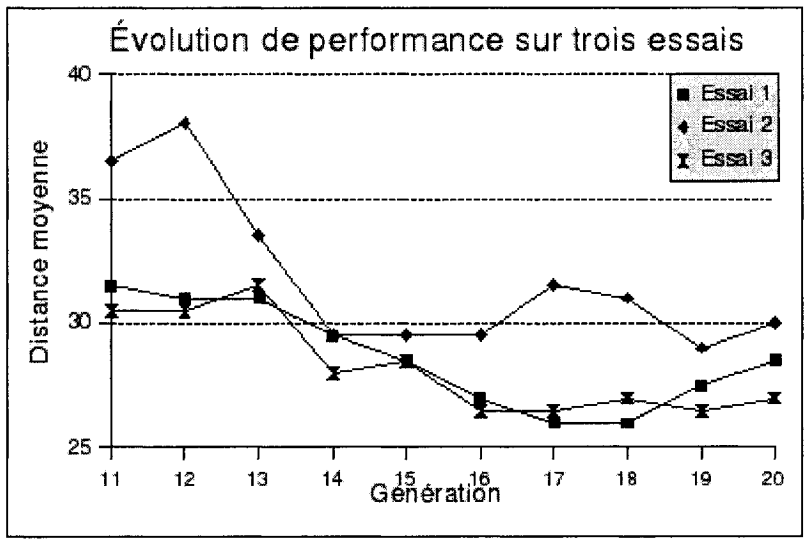

FIG. 4.8 Résultats de la deuxième évolution

En comparant les figures 4.7 et 4.8 on remarque que dans cette deuxième évolution, les gains sont moins prononcés et moins fréquents. Les trois contrôleurs atteignent chacun un minimum local dans les dernières générations.

TAB. 4.9 Poids synaptiques (en mV) pour la direction après la deuxième évolution

Neurone gauche				Neurone droite			
Senseur	E. 1	E. 2	E. 3	Senseur	E. 1	E. 2	E. 3
Droite	524	640	431	Gauche	385	492	489
+40°	222	276	379	-40°	570	294	194
+120°	302	469	679	-120°	647	582	373
Prox.	175	157	139	Prox.	-	-	204
Lent	152	140	-	Lent	189	220	-

TAB. 4.10 Poids synaptiques (en mV) pour la vitesse après la deuxième évolution

Neurone accélération				Neurone décélération			
Senseur	E. 1	E. 2	E. 3	Senseur	E. 1	E. 2	E. 3
Droite	-	273	117	+120°	153	130	181
Gauche	123	189	113	-120°	103	161	-
Lent	242	257	225	Prox.	124	196	-
-120°	100	-	-	Vite	285	181	368
n/a	-	-	-	-40°	100	-	-

Les tables 4.9 et 4.10 montrent les poids synaptiques du CE gagnant avec la meilleure performance de chaque essai. Encore ici, pour les poids qui sont inférieurs à 100mV, on dit que la connexion synaptique est inutile, et on marque ceci par un tiret. Le contôleur le plus performant est celui de la 18ième, 19ième et 20ième géneration, respectivement, pour les essais 1, 2 et 3.

Une description qualitative des comportements de ces trois contrôleurs est donnée à la table 4.11. En examinant les comportements des robots obtenus, et en les comparant avec ceux de la table 4.8, on observe quelques faits intéressants.

En premier, à l'exception de l'essai 2, les comportements après 10 générations et après 20 générations demeurent semblables. C'est intuitif que plus un contrôleur devient spécialisé, moins il est probable qu'il va changer drastiquement de stratégie.

Quant à l'essai 2, on peut probablement attribuer son changement de stratégie au fait qu'il souffre de fortes pénalités en performance dans les générations 11 et 12 (voir la figure 4.8). À ce point, les enfants gagnants étaient probabablement plus différents qu'améliorés, ce qui pourrait expliquer cette pénalité en performance et le changement de stratégie. Dans les générations suivantes, on voit que le contrôleur devient de plus en plus raffiné.

En deuxième, on remarque qu'il-y-a plusieurs stratégies distinctes qui donnent des performances semblables. Par exemple, le robot de l'essai 2 encercle la cible à haute vitesse tandis que ceux des autres essaient de la suivre à sa vitesse. En gardant toujours sa vitesse élevée, le robot de l'essai 2 a l'avantage de pouvoir rapidement rattrapper la cible après une perte de contact, comme présenté à la figure 4.9. Par contre, le rayon de ses encerclements est plus grand que la distance typique à laquelle les deux autres robots suivent la cible. Un autre exemple est la distance que ces robots essaient de garder entre eux et la cible. L'essai 3 reste plus éloigné de la cible que l'essai 1, ce qui réduit sa performance, mais il ne dépasse et perd rarement la cible - un fort avantage. Quand le robot de l'essai 1 perd contact avec la cible, ça lui prend du temps à la rattrapper, au détriment de sa performance. Évidemment, le compromis entre minimisation de la distance de poursuite et continuité de la poursuite est très important dans le calcul de la distance moyenne.

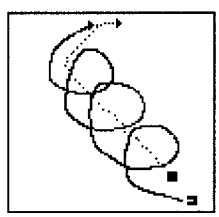

FIG. 4.9 Comportement du robot de l'essai 2 (solide) après 19 générations

TAB. 4.11 Description qualitative des comportements des robots après la deuxième dizaine de générations

Contrôleur	Déscription	Dist. Moy.
Essai 1	Vitesse très stable, reste proche derrière la cible mais la dépasse périodiquement. Efficace mais pas pressé à rattrapper la cible.	28.5
Essai 2	Vitesse plus élévée que la cible, l'encercle et la dépasse fréquement. Forte préférence de tourner à droite.	30
Essai 3	Vitesse très stable, reste plus distant derrière la cible que Essai 1, mais la dépasse rarement.	27

4.3.3 Optimisation des poids synaptiques pour trouver le local minimum

On essaie maintenant de vérifier si les contrôleurs de la deuxième évolution sont près d'un minimum local, et si oui, c'est quoi la plus faible distance moyenne qu'ils peuvent obtenir. Dans la figure 4.8 on a noté que dans les dernières générations, les performances s'amélioraient peu ou pas du tout. Il est fortement probable que les contrôleurs approchent chacun un local minimum qui limite leur performance. Dans le but de "creuser" dans ce local minimum, on prend le meilleur contrôleur de chaque essai, et on l'optimise avec des défauts beaucoup plus faibles sur les poids synaptiques. La table 4.12 liste les paramètres de l'évolution. On augmente le nombre d'itérations par génération à un million pour assurer que les distances moyennes obtenues par les CE sont répétables.

Dans chaque cas, les performances augmentent très peu, et même pas du tout. La table 4.13 montre les performances des contrôleurs avant et après les trois générations.

TAB. 4.12 Paramètres de la troisième évolution

	Valeur
Nombre de générations	3
Itérations par génération	1 000 000
Enfants par génération	12
$Facteur_{erreur}$ de poids	approx. 3

TAB. 4.13 Résultats de la troisième évolution

	Performance initiale	Peformance finale
Essai 1	26	25
Essai 2	29	29
Essai 3	26	24

On conclut que les quatres contrôleurs sont chacun dans un minimum local. Étant donné qu'on n'a cherché que dans une petite partie de l'espace de solutions, il est fortement improbable qu'une distance moyenne de 24 pixels soit le minimum global. En fait, à la prochaine section, on verra que des contrôleurs encore plus performants existent. Si on veut améliorer davantage la performance des trois contrôleurs obtenus, il suffirait probablement de produire quelques générations avec des mutations plus importants sur les poids synaptiques avant de recommencer la procédure de cette section. Ceci aura l'effet d'éloigner chaque contrôleur de son minimum local, avant de commencer à rechercher un autre minimum local qui sera meilleur.

4.3.4 Notre meilleur contrôleur

Dans cette section on présente un contrôleur de robot qui suit la cible mobile à une distance moyenne de 19 pixels. La procédure pour l'obtenir n'est pas bien

TAB. 4.14 Poids synaptiques (en mV) du contrôleur le plus optimisé, avec changements depuis le contrôleur initial entre paranthèses

	Neurone			
Senseur	Gauche	Droite	Accélère	Décélère
Droite	433 (+33)	-	146 (-54)	-
+120°	691 (+401)	-	-	232 (+22)
+40°	285 (+55)	-	-	-
-40°	-	292 (+62)	-	-
-120°	-	418 (+128)	-	265 (+55)
Gauche	-	309 (-91)	- (!)	-
Prox.	302 (+172)	- (!)	-	357 (+207)
Lent	199 (+79)	489 (+369)	524 (+334)	-
Vite	-	-	-	198 (+18)

documentée[8], mais c'est toujours un descendant du contrôleur de la section 4.1. Il a fallu produire plus d'une trentaine de générations avec des magnitudes de défauts semblables à ceux qu'on a utilisé dans les deux sections précédentes pour l'obtenir. La table 4.14 liste ses poids synaptiques. Les changements des poids synaptiques depuis le contrôleur de la section 4.1 sont indiqués entre parenthèses (un point d'exclamation indique que la synapse est rendu inutile). Tous les autres paramètres sont identiques à ce contrôleur initial.

Avec ce contrôleur, le robot est très habile à suivre la cible mobile de très près et de reproduire sa vitesse. Ses virages sont rapides et bien contrôlées, même à basse vitesse, sans accélération excessive. S'il perd contact avec la cible (ce qui est rare), le robot tourne et accélère suffisamment pour la rattrapper rapidement, et décélère assez pour ne pas la dépasser. C'est impressionant à voir !

En examinant les poids synaptiques à la table 4.14, on remarque qu'il n'y a peu de symmétrie (par contre, le réseau de neurones initial était parfaitement symmétrique

[8]La grande part de l'optimisation a été réalisé, en s'amusant, par un artiste, Chan Tchen

quant à la direction). Deux asymmétries notables sont le fait que le robot est fortement encouragé à tourner à gauche en proximité à la cible, et que c'est fortement encouragé de tourner à droite quand il roule lentement. Quand le robot suit bien la cible (c'est à dire qu'il roule relativement lentement derrière la cible en forte proximité), ces deux effets se balancent, et le robot n'a pas de bias en direction (autre qu'une décalage à droite - voir le paragraphe suivant). Par contre, quand le robot roule lentement à distance de la cible, il tourne à droite pour le retrouver, ce qui donne un effet secondaire d'augmenter son accélération (à cause du synapse du neurone d'accélération branchée au senseur de direction droite), comme présenté à la figure 4.10a. Quand le robot roule trop vite en proximité de la cible (typiquement après l'avoir rattrappée), il tourne à gauche pour éviter de la dépasser et pour ralentir (il n'y a pas de synapse du neurone d'accéleration branchée au senseur de direction gauche), comme présenté à la figure 4.10b.

Deux autres asymmétries notables sont le fait que robot est encouragé à suivre la cible décalée à sa droite, et qu'il tourne plus rapidement à gauche quand il dépasse la cible. Ces comportements sont présentés à la figure 4.10c et d. Ils ont l'effet combiné de faire que quand le robot dépasse la cible, il la dépasse typiquement à droite. Ensuite, la cible se retrouve derrière et à gauche du robot (détecté par le senseur -120) ce qui lui encourage de tourner rapidement à gauche pour la rattrapper (une stratégie qui marche très bien). Dans le cas où la cible se retrouve derrière et à droite du robot, ce qui arrive quand le robot tourne à gauche pour ralentir derrière la cible (voir la figure 4.10b), le robot est toujours encouragé de tourner à droite par sa synapse sur le senseur -120, mais moins fortement. Ceci probablement parce qu'il a déjà un bias pour se décaler à droite de la cible. En tout cas, les asymmétries du réseau contribuent à la forte performance du robot en poursuite de la cible mobile.

De manière générale, des poids synaptiques asymmétriques peuvent donner des avantages en performance. Le réseau de neurones qu'on vient de présenter profite

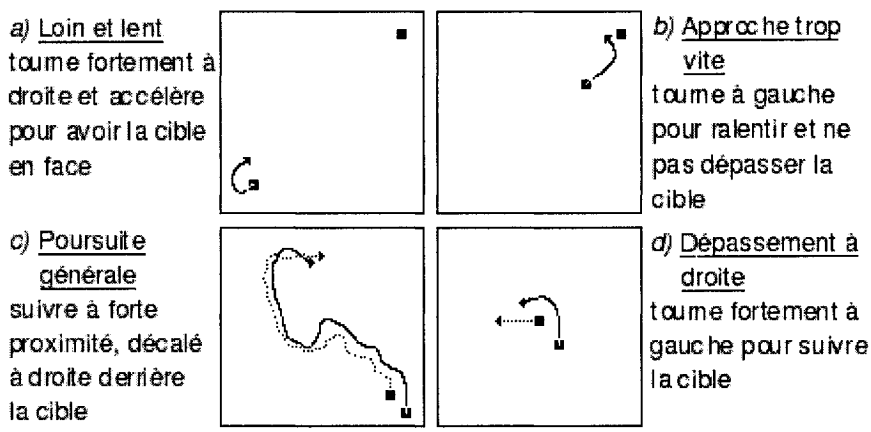

FIG. 4.10 Comportement du robot final

du fait de pouvoir se comporter différemment quand la cible est à droite ou à gauche, quand le robot tourne à droite ou à gauche, etc. Par contre, le réseau de neurones initial (section 4.1), ayant des poids synaptiques symmétriques, ne pourrait pas en profiter de ce phénomène, ce qui pourrait limiter sa performance.

4.3.5 Sommaire de l'optimisation par évolution

Il existe de très bons contrôleurs comme celui qu'on vient de présenter, qui suit la cible à une distance moyenne de 19 pixels. On est surpris de constater qu'il est possible d'obtenir un contrôleur aussi performant étant donné l'effort nécessaire pour produire, manuellement par intuition, le contrôleur de la section 4.1. Ce contrôleur n'obtient qu'une distance moyenne de 44 pixels. L'application évolutive fut indispensable pour obtenir des contrôleurs plus performants, et ceci en très peu de temps[9].

Dans les sections 4.3.1, à 4.3.3 on a détaillé l'exploration d'une petite partie de l'espace de solutions dans le but de produire des contrôleurs optimisés. La recherche

[9]Les évolutions faites dans ce chapitre ne durent que quelques minutes par 100 000 itérations sur un ordinateur personnel

d'un minimum global à ce problème pourrait être le sujet d'un autre mémoire au complet ! En se servant de l'application évolutive, on a produit, en relativement peu de temps, plusieurs robots optimisés ayant des comportements distincts. En plus, ces comportements nécessaires pour réduire la distance moyenne ne sont pas toujours évidents : la stratégie des robots des sections 4.1 et 4.3.4 diffèrent beaucoup.

Le robot de la section 4.3.4 est très habile à suivre la cible mobile. En fait, on ne pense même pas être capable de battre cette performance en contrôlant personnellement le robot ! Ceci valide notre preuve de concept : les circuits de neurones à impulsions peuvent servir comme contrôleur embarqué.

4.4 Conclusion

Dans ce chapitre on a présenté la conception, par essai-erreur, d'un réseau de neurones qui donne au robot une performance satisfaisante. Ensuite, on a présenté une application évolutive qui sert a "reproduire" des réseaux de neurones, avec erreurs, dans le but d'améliorer leur performance. En se servant de cette application, on a obtenu plusieurs réseaux de neurones à comportements distincts qui permettent le robot de suivre la cible de très près. Ayant validé notre preuve de concept, on présente le potentiel des circuits de neurones à impulsions ainsi qu'un sommaire de leurs avantages et désavantages au chapitre suivant.

CHAPITRE 5

APPLICATIONS AVANCÉES ET COMPTE RENDU

Dans ce chapitre, on applique les circuits de neurones à impulsions sur le problème du OU-exclusif pour mieux les comparer avec d'autres technologies et types de neurones artificiels. Ensuite, on décrit des variations de la tâche "suivre une cible mobile" qui pourraient servir à tester davantage ces architectures de neurones en plus de profondeur. Ensuite, on présente une architecture de puce reconfigurable de neurones à impulsions qui pourrait rentabiliser l'implantation en matériel de ce type de neurones. Et finalement, on énumère les avantages et désavantages des neurones à impulsions.

5.1 Le OU-exclusif

5.1.1 Intérêt du problème du OU-Exclusif

Le OU-exclusif est un problème logique standard qui est souvent utilisé pour comparer différents types de neurones, réseaux ou technologies. La table de vérité et une illustration du OU-Exclusif sont présentées à la figure 5.1. Cette fonction est un problème intéressant parce que sa solution n'est pas linéairement séparable. C'est à dire qu'il est impossible de diviser les sorties qui sont "1" de ceux qui sont "0" par une seule ligne. Il prend, par contre, deux lignes pour diviser les sorties opposées (voir la figure 5.1).

On présente le problème du OU-Exclusif pour deux raisons. Premièrement, on montre que les neurones à impulsions sont capables de résoudre ce problème de

X	Y	Sortie
0	0	0
0	1	1
1	0	1
1	1	0

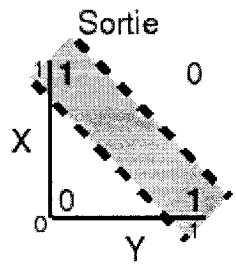

FIG. 5.1 Table de vérité et illustration du OU-Exclusif

manière intéressante, et on compare la solution à celle d'autres technologies. Et, deuxièmement, on discute les implications de pouvoir résoudre un tel problème pour la situation décrite dans les chapitres précédents.

5.1.2 Implantation du OU-Exclusif

On décrit à la figure 5.2 un réseaux de neurones à impulsions de type *leaky integrate-and-fire* capable d'implanter le OU-exclusif.

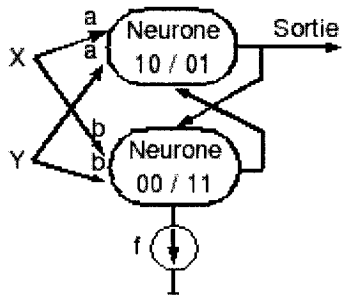

FIG. 5.2 Implantation possible du OU-Exclusif

Cette implantation du OU-Exclusif ne requiert que deux neurones dans une seule couche. Ses entrées sont numériques (0.0V représente un 0-logique et 0.9V représente un 1-logique) et sa unique sortie représente un 1-logique par des impulsions et un 0-logique par leur absence complet.

Les deux neurones sont dans un regroupement le-gagnant-prend-tout. De manière

générale, dans chacun des quatres cas possibles sur les entrées, la somme des courants synaptiques décide quel neurone gagnera. Le neurone *00 / 11* a une fuite représentée par f. Dans la figure 5.2, a et b représentent le courant dégagé par la synapse quand son entrée est 0.9V (1-logique). Pour que le Neurone *01 / 10* gagne quand une seule entrée est positive (X ou Y = 1), il faut que ce courant satisfaisse l'équation 5.1. Pour que le neurone *00 / 11* gagne quand les deux entrées sont positives (X et Y = 1), il faut que l'équation 5.2 soit satisfaite. Quand les deux entrées sont zéro (X et Y = 0), il n'y a pas de courant sortant des synapses, et aucun neurone pulse. En fusionnant les deux équations, on obtient l'équation 5.3 qui dicte comment choisir les poids synaptiques et la fuite.

$$a > b - f \tag{5.1}$$

$$2a < 2b - f \tag{5.2}$$

$$b - f < a < b - \frac{f}{2} \tag{5.3}$$

En se servant de la table 2.1 et/ou la figure 1.3, il est possible de trouver un nombre infini de combinaisons qui satisfont l'équation 5.3. La table 5.1 donne une combinaison qui produit le résultat désiré. Le fonctionnement du circuit avec ces paramètres est décrit par la table 5.2. Quand le neurone *01 / 10* gagne, son courant somatique est 390nA, ce qui donne une fréquence d'impulsions de 16.7Hz (voir la figure 1.8). On remarque que les différences entre les courants somatiques des deux neurones est faible dans les cas où au moins une entrée est positive. Ceci indique qu'il n'y a peu de tolérance pour des erreurs dans les paramètres ou les tensions d'entrée. Pour rendre le circuit plus robuste, il suffit de choisir des paramètres dont la fuite est plus élevée. La pénalité sera cependant une plus grande consommation de puissance.

TAB. 5.1 Paramètres pour l'implantation du OU-exclusif

Paramètre	Valeur	Implication
a : Poids synaptique (01 / 10)	142mV	a = 390nA
b : Poids synaptique (00 / 11)	150mV	b = 465nA
f : Fuite (00 / 11)	100nA	f = 100nA

TAB. 5.2 Fonctionnement de l'implantation du OU-exclusif

X	Y	$I_{synaptique01/10}$	$I_{synaptique00/11}$	Sortie
0	0	0nA	0nA	0V DC
0	1	390nA	365nA	16.7Hz
1	0	390nA	365nA	16.7Hz
1	1	780nA	830nA	0V DC

5.1.3 Comparaison aux autres technologies

Une porte OU-exclusif à deux entrées en logique numérique peut être réalisé avec très peu de transistors avec la technologie CMOS, comme présentée à la figure 5.3[1]. Par contre, l'implantation en neurones à impulsions qu'on vient de présenter nécessite 31 transistors (et 8 condensateurs). Quant aux implantations neuronales formels, le OU-exclusif peut être résolu avec trois perceptrons : deux dans une couche cachée et un dans la couche de sortie (Touretzky et Pomerleau, 1989). Une seule couche de perceptrons n'est pas capable de résoudre le problème, tandis-que dans que dans l'implantation présentée, on le résoud avec deux neurones à impulsions dans une seule couche. Donc, notre réseau présenté à la figure 5.2 compare favorablement avec celui des perceptrons en termes d'architecture et nombre de neurones. Quant au nombre de transistors, il est du même ordre de grandeur qu'une implantation "force-brute" en CMOS. Même si le OU-exlusif est un problème simple, son implan-

[1]Dans chaque implantation on suppose qu'on a accès aux entrées et leur inverse

tation n'est pas toujours évident. Ici on a montré que c'est possible de concevoir une solution élégante et efficace en circuits de neurones à impulsions compétitifs (LGPT).

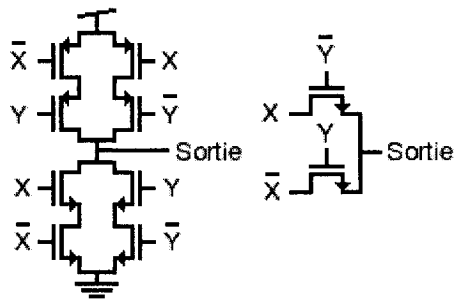

FIG. 5.3 Implantations possibles du OU-Exclusif en CMOS

5.1.4 Implications pour l'intelligence artificielle

En plus d'être un problème standard pour la comparaison de réseaux de neurones, le OU-exclusif représente aussi un outil important pour les cerveaux biologiques et artificiels. Par exemple, sa négation, le non-OU-exlusif, représente l'équivalence logique (X *if-and-only-if* Y). Ceci permet le calcul des conditions nécessaires et suffisantes pour une action (Churchland et Sejnowski, 1992).

Quant au robot présenté, le OU-exclusif pourrait servir à permettre de choisir entre des buts mutuellement exclusifs. Par exemple, le robot pourrait choisir entre la poursuite de la cible ou la fuite d'un prédateur, dans le cas où il y aurait un prédateur (voir section 5.2.1).

5.2 Avancement de la tâche du robot

La tâche de chasser une cible mobile est relativement simple et ne représente qu'un seul but. C'est pour ces raisons et ceux présentés au chapitre 3, qu'on l'a utilisée

comme application embarquée pour notre preuve de concept des circuits de neurones à impulsions de type *leaky integrate-and-fire*. Pour explorer les capacités de ces neurones plus en profondeur, il faudrait les appliquer sur des tâches de plus en plus avancées. Dans la section précédente, on a vu que ces neurones étaient capables d'implanter le OU-exclusif. Est-ce-qu'ils pourraient donner des comportements plus avancées au robot et lui permettre d'accomplir des tâches plus difficiles ?

Dans la prochaine section, on présente le scénario de prédateur et proie qui peut aussi être exploré dans notre application de développement et de simulation. Ensuite, on présente d'autres tâches avancées pour le robot qui pourraient être explorées après de légères modifications de la mesure de performance déjà proposée (c'est à dire des changements dans le code source).

5.2.1 Prédateur et proie

Une tâche avancée pour le robot qui est possible avec les outils présentés est l'évasion d'un prédateur en même temps que la chasse de la proie (la cible mobile). Dans l'application de simulation et de développement, il est possible de munir le robot avec des senseurs pour détecter le véhicule de référence et de le traiter comme prédateur. Les statistiques dans l'application de développement et simulation indiquent la distance moyenne entre le robot et ce prédateur. Le but du robot devient donc de maximiser sa distance au prédateur tout en minimisant sa distance à la cible mobile.

Le réseau à deux couches présenté à la figure 5.4 a été conçu, par essais et erreurs, pour rapprocher le robot à la cible, et de l'éloigner du prédateur. La première couche de neurones consiste em deux regroupements LGPT : le premier décide la direction nécessaire pour poursuivre la cible (*Pleasure*) et le deuxième décide la

direction nécessaire pour éviter le prédateur (*Pain*). La deuxième couche nécéssite un seul regroupement LGPT qui décide quelles impulsions de la première couche seront transmises aux actuateurs. Cette couche favorise légèrement la poursuite de la cible mobile et inclut aussi des entrées provenant des senseurs de direction inverse comme rétroaction négative. Le réseau n'affecte pas la vitesse du robot (le but de cet exemple est de montrer que la tâche est réalisable plutôt que de développer un réseau optimal).

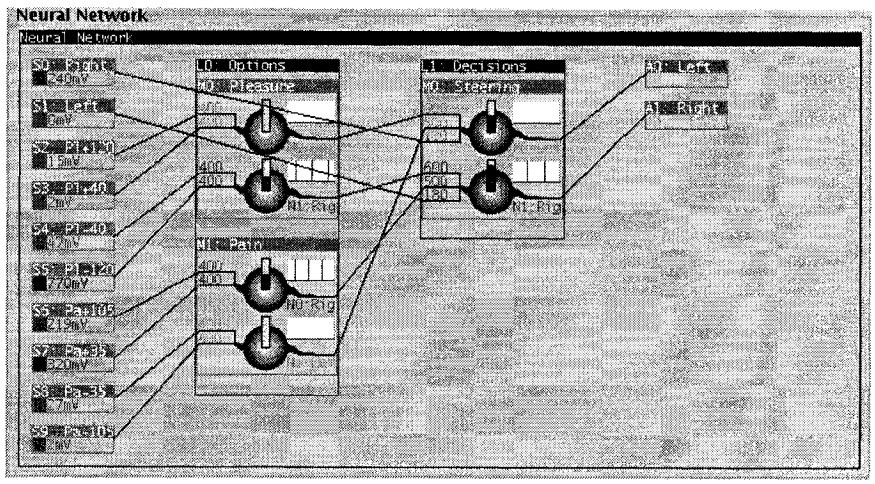

FIG. 5.4 Réseau qui effectue la tâche de prédateur/proie

La performance de ce réseau de neurones est donnée à la table 5.3. La distance moyenne entre le robot et la cible est presque la moitié de la distance moyenne entre le robot et la cible. Il serait intéressant de continuer la recherche sur ce problème pour mieux évaluer les circuits de neurones à impulsions de type *leaky integrate-and-fire*. La performance du robot pourrait être amélioré par évolution avec erreurs, et des OU-exclusif pourraient être intégrés dans le réseau pour déterminer s'ils améliorent la performance ou réduisent la taille du réseau, par exemple.

TAB. 5.3 Performance du contrôleur qui évite aussi un prédateur

Paramètre	Distance (pixels)
Robot et cible	78
Robot et prédateur	144
Prédateur et cible	127

5.2.2 Autres possibilités

D'autres tâches légèrement plus avancées possibles à explorer incluent les suivantes :

- poursuite de la cible mobile à faible consommation d'énergie (pénalité pour les fortes accélérations/virages)
- rattrapper la cible mobile dans un délai minimum (temps nécessaire pour arriver à une certaine distance de la cible)
- poursuite de la cible mobile à une distance prescrite (pénalité pour être trop proche ou loin)
- etc.

L'application de développement et de simulation, comme tel, n'est pas configuré pour calculer et afficher ces mesures de performances alternatives, mais son code pourrait facilement être modifié pour le faire.

5.3 Implantation matériel du circuit

Les circuits de neurones à impulsions présentés peuvent, évidemment, être implantés dans une puce. Par contre, il y a deux inconvénients à cette fin : le coût de production d'une puce et l'impossibilité de reconfigurer les paramètres du circuit. En d'autres mots, il ne fait aucun sens économique de fabriquer une puce pour quelques robots dans une laboratoire (spéciallement si un seul paramètre er-

TAB. 5.4 Paramètres configurables du circuit

Paramètre	Déterminé par	Prévalence
Poids synaptique	Tension analogique	1 / synapse
Fuite somatique	Tension analogique	1 / neurone
Seuil somatique	Capacité	1 / neurone
Regroupement LGPT	Signal numérique	1 / pair de neurones

roné peut gravement nuire à leur fonctionnalité!). Pour éviter ces inconvénients, il faudrait une puce reconfigurable produit en grande série pour minimiser le coût à l'unité. Dans cette section on présente une architecture de puce qui pourrait peut-être un jour répondre à ces besoins.

La table 5.4 présente les paramètres configurables dans un circuit de neurones à impulsions de type *leaky integrate-and-fire*. Par exemple, pour modifier un poids synaptique il suffit de modifier une tension analogique. Modifier le seuil somatique veut dire changer une capacité. Mais ce dernier peut aussi être contrôlé par un réseau de condensateurs numériquement configurables (*programmable capacitor array*, donc des signaux numériques (voir figure 5.5[2]). Tout pour dire que les paramètres configurables du circuit peuvent être déterminées par des tensions analogiques et numériques.

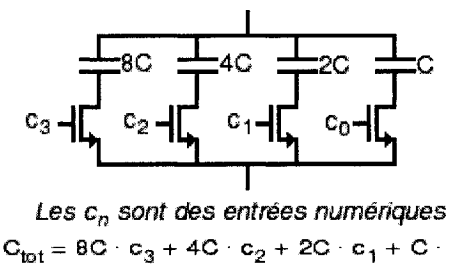

FIG. 5.5 Réseau de condensateurs numériquement configurables à 4-bits

Un circuit reconfigurable de neurones à impulsions aurait les propriétés suivantes :

[2]Ce circuit pourrait remplacer le condensateur 8nF du circuit somatique (voir la figure 1.6)

- interface de communication avec un ordinateur personel (pour le changement et la lecture des paramètres)
- mémoire pour sauvegarder les paramètres
- ports parallèles numériques internes pour les paramétres numériques
- convertisseur numérique-analogique (CNA) pour les paramètres analogiques

Une manière simple d'implanter ces propriétés est d'intégrer un microcontrôleur et un CNA dans la puce. Il existe une variété d'architectures de microcontrôleurs 4 et 8 bit qui peuvent communiquer sériellement ou par USB avec un ordinateur personel, qui ont une mémoire non-volatile (EEPROM) intégrée, et qui ont des ports parallèles numériques. Le microcontrôleur pourrait communiquer avec le CNA par interface parallèle ou sérielle. L'architecture d'une telle puce comprenant quatre neurones avec dix synapses chacun et seuil somatique variable à 4-bits est présentée à la figure 5.6.

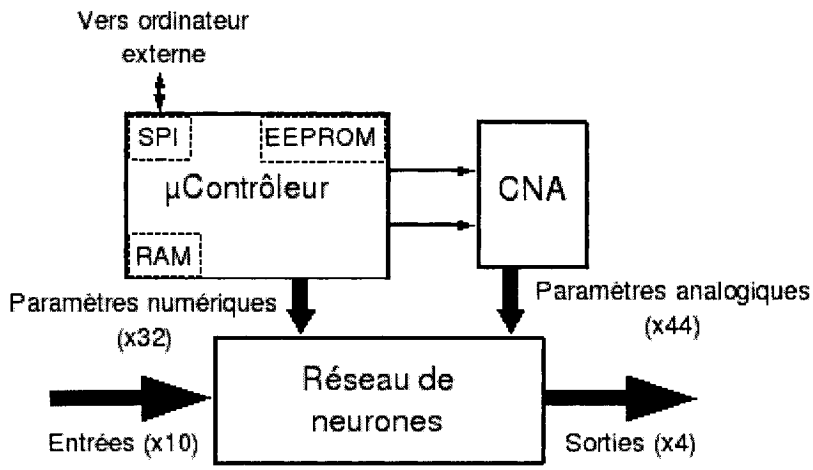

FIG. 5.6 Hierarchie de la puce

Avec une puce reconfigurable, simple à utiliser et économique, disponible en grand nombre, l'utilisation de circuits de neurones à impulsions de type *leaky integrate-and-fire* pour des applications où leurs avantages sont clairs serait fortement encouragée. Des réseaux multi-couches avec plus que quatres neurones pourraient être réalisés simplement en branchant plusieurs puces ensemble, comme présenté à la

figure 5.7.

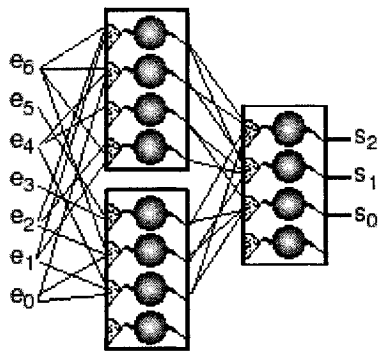

FIG. 5.7 Réseau à deux couches et 10 neurones implanté avec trois puces

5.4 Avantages et désavantages

Dans cette section on énumère les avantages et les désavantages des neurones à impulsions de type *leaky integrate-and-fire* en ITGÉ analogique et les contrôleurs neuronales de robot présentées dans le chapitre 4.

5.4.1 Avantages des circuits de neurones à impulsions

Un avantage des neurones à impulsions de type *leaky integrate-and-fire* en ITGÉ analogique est leur compacité. Un neurone avec quatre synapses, comme ceux des réseaux du chapitre 4, nécessite 24 transistors et 6 condensateurs. C'est à dire que ces réseaux de 4 neurones, n'ont qu'une centaine de transistors. Un circuit intégré avec cent transistors ou moins est désigné *intégration à petite échelle* (IPÉ). Comme comparaison, les circuits numériques en IPÉ, n'implantent que quelques portes numériques (ET, OU, OU-excusif...). Il est difficile d'imaginer pouvoir contrôler le même robot qu'avec quelques portes logiques et d'obtenir la même performance que le réseau de neurones de la section 4.3.4 !

Un avantage des neurones à impulsions de type *leaky integrate-and-fire* en ITGÉ analogique est leur flexibilité. Comme on a vu au chapitre 4, une seule architecture a pu donner au robot une variété de comportements distincts et intéressants - et ceci en ajustant que les poids synaptiques.

Un avantage des circuits de neurones présentés dans le chapitre 1 est le fait qu'ils acceptent naturellement des entrées numériques et analogiques. Comme résultat, le même circuit synaptique peut être utilisé pour brancher un neurone aux deux types d'entrée. Ceux-ci incluent les sorties des autres neurones à impulsions, des signaux de modulation d'impulsions en durée, et des senseurs binaires (numériques), ou des senseurs analogiques, sinusoïdes et des tensions constantes (analogiques). La puce présentée dans la section 5.3 serait donc très versatile, sans besoin de circuits supplémentaires pour transformer différents types d'entrées.

Les circuits de neurones à impulsions de type *leaky integrate-and-fire* sont une hybride d'éléments numériques et analogiques, et profitent des avantages de chacun. La multiplication synaptique et l'intégration somatique sont réalisées en circuits analogiques compacts et efficaces (ils font ces deux fonctions de manière plus naturelle que les circuits numériques). Les impulsions émises par chaque neurone sont numériques et peuvent être transmises à distance avec peu de sensibilité au bruit (les signaux analogiques sont plus sensibles au bruit). Et finalement, un avantage des réseaux de neurones implantés en matériel, et non simulés sur un ordinateur sériel, est le fait qu'ils fonctionnent en parallèle.

5.4.2 Désavantages des circuits de neurones à impulsions

Un désavantage de ces circuits de neurones à impulsions est le fait qu'il faut les implanter dans une puce pour les utiliser et tester réellement. Dans ce mémoire, on

n'a pu confirmer les résultats de la simulation à cause du coût élevé de production d'une puce.

Un désavantage potentiel des circuits présentés dans le chapitre 1 est le fait qu'ils sont plutôt optimisés pour la compacité que pour la faible consommation de puissance ou le réalisme (de manière générale, ces trois sont mutuellement exclusifs). Pour des applications qui demandent, par exemple, une faible consommation de puissance au détriment de la compacité, il faudrait utiliser des circuits modifiés. L'impact étant que l'application de développement et de simulation et l'application évolutive ne seraient plus valides à moins que leurs modèles de comportements des neurones soient adaptés pour conformer aux nouveaux circuits.

Un désavantage de base de l'infrastructure d'optimisation présentée dans le chapitre 4 est le fait que ça ne génère pas nécessairement des réseaux de neurones optimaux selon une mesure de performance donnée. L'application évolutive implantée a bien réussi à produire des contrôleurs performants dans le chapitre 4, mais ce n'est pas nécessairement la méthode la plus vite, ni efficace, ni robuste. Il serait souhaitable de poursuivre la recherche pour trouver de meilleurs algorithmes d'optimisation pour les circuits de réseaux de neurones à impulsions.

5.5 Une recherche courante comparable

En 2006, à la conclusion de cette recherche, Dario Floreanu et son équipe ont publié un article détaillant une recherche comparable (Floreanu et al., 2006). Dans leur recherche, un robot Khepera avec un "cerveau" de neurones à impulsions doit naviguer dans un environnement fermé sans toucher aux murs (voir figure 5.8). Les neurones à impulsions sont modélisés à l'aide d'un logiciel implanté sur un microcontrôleur peu dispendieux. L'équipe a optimisé avec succès leurs réseaux de

neurones par un algorithme évolutif.

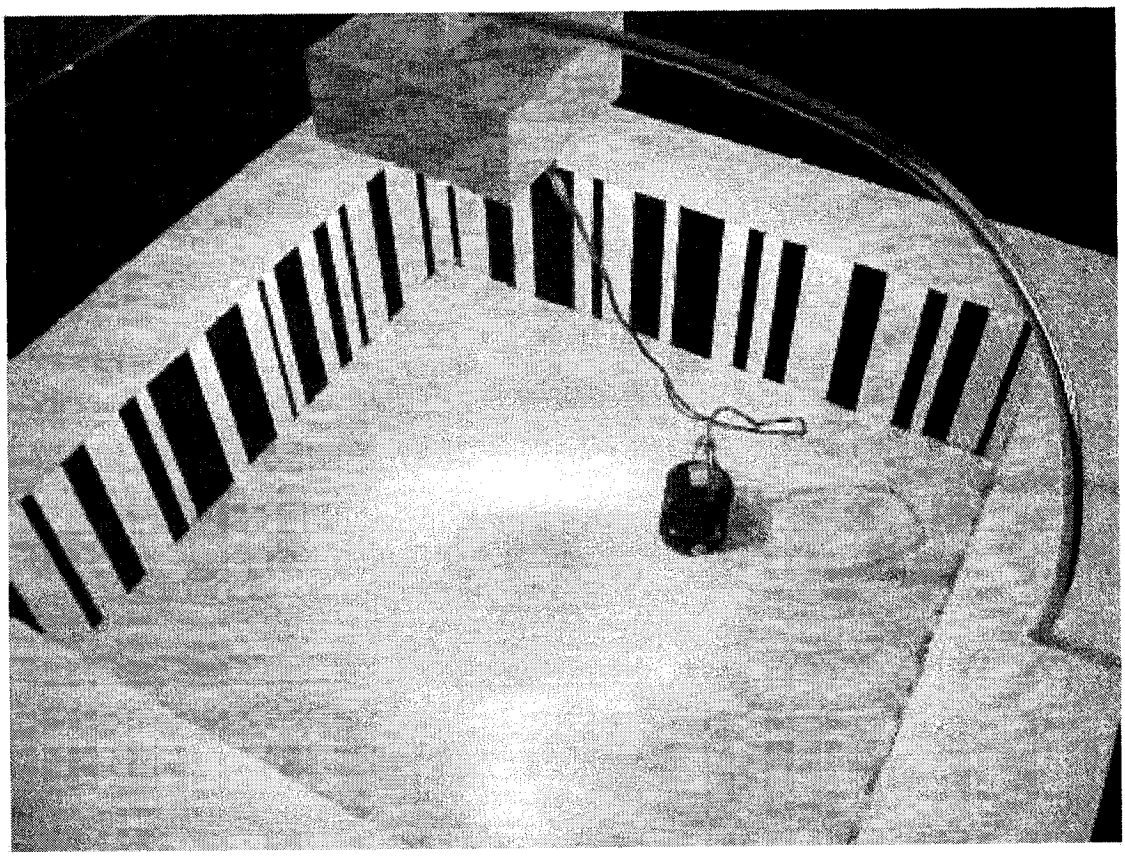

FIG. 5.8 Robot Khepera de l'équipe de Floreanu dans son environnement

Les deux approches (la leur et la notre) cherchent à implementer des neuro-contrôleurs minimales. Leurs neurones ultra-simplifiées et linéaires exploitent presque optimalement les resources d'un microcontrôleur. Les notres utilisent un minimum de transistors. Ils avouent que leurs neurones consomment plus de puissance et calculent à une vitesse et un niveau de parallelisme inférieur qu'une implantation en ITGÉ analogique. Par contre, nous sommes limités à faire des simulations en l'absence d'une puce reconfigurable de neurones *integrate-and-fire.*

Le fait que Floreanu et son équipe utilisent un algorithme évolutif pour entraîner leurs neuro-contrôleurs (qui ont moins de paramètres que les notres) confirme la complexité de la tâche d'optimisation, même pour des architectures minimales.

L'optimisation par algorithme évolutif est donc une technique avantageuse.

Finalement, une différence à souligner est le fait que le neuro-contrôleur qu'on présente est le seul élément électronique entre les senseurs et actuateurs du robot. Les signaux des senseurs sont les entrées des circuits synaptiques, et les impulsions des neurones contrôlent directement des moteurs à pas. Quant au robot de l'équipe de Floréanu, les signaux optiques sont normalisés et passent par un filtre de Laplace avant de devenir les entrées des neurones, et les "impulsions" des neurones de sortie sont moyennées pour déterminer la vitesse de chaque roue.

5.6 Sommaire

Dans ce mémoire nous avons montré qu'un cerveau artificiel composé de circuits de neurones à impulsions de type *leaky integrate-and-fire* est capable de contrôler un simple robot de manière intéressante. Nous avons donc validé notre preuve de concept en montrant que des circuits de neurones à impulsions peuvent servir comme contrôleur embarqué.

Dans la poursuite de nos objectifs, nous avons mis en place une infrastructure de logiciels pour le développement, simulation et évolution de circuits de réseaux de neurones à impulsions. Ces outils sont librement disponibles et peuvent servir à d'autres chercheurs pour une variété d'applications.

Le robot simulé effectue une tâche inspirée de la biologie : la poursuite d'une cible mobile, ce qui est analogique à la poursuite d'une proie, d'un partenaire, etc. Cette tâche est bien adaptée à ses capacités. Le réseau de neurones représente l'unique contrôleur entre ses senseurs et ses actuateurs. Le robot pourrait facilement être implanté en matériel.

Le robot et son réseau de neurones peuvent être évolués avec erreurs pour obtenir de meilleure performance. Nous avons évolué le réseau de neurones pour produire un contrôleur de robot plus performant qu'un contrôleur humain donné.

Nous avons conçu d'autres réseaux pour résoudre le problème du OU-exclusif et le scénario de prédateur/proie. Dans chaque cas, nous avons profité des regroupements le-gagnant-prend-tout configurables qu'on a présenté à la section 1.6.

Nous avons présenté l'architecture d'une puce de circuits de neurones à impulsions reconfigurables. Une telle puce pourrait rentabiliser l'utilisation de ces circuits dans une grande variété d'applications embarquées.

Les circuits de neurones à impulsions de type *leaky integrate-and-fire* sont compacts : le réseau de neurones du robot ne nécessite qu'une centaine de transistors. En plus, ce réseau est flexible, accepte deux types de signaux sur ses entrées (analogique et impulsions), et fonctionne en parallèle.

5.7 Conclusion

Les neurones à impulsions électroniques profitent des avantages des circuits numériques et analogiques. Ils peuvent être organisés en regroupements le-gagnant-prend-tout numériquement configurables pour obtenir des comportements souhaitables. Avec les logiciels développés dans cette recherche, et librement accessibles par Internet, il est possible de faire du prototypage rapide de neurocontrôleurs à pulsations, tout en visualisant leur comportement. Un circuit de neurones à impulsions est capable d'interfacer directement avec des senseurs et des actuateurs tels que les senseurs analogiques et les moteurs à pas du robot suiveur présenté. L'optimisation d'un neurocontrôleur à impulsions par algorithme évolutif est souhaitable, étant donné la complexité de cette tâche. Un circuit de quatre neurones

à impulsions optimisé permet à un simple robot de suivre une cible mobile, en moyenne, de plus près qu'un contrôleur humain entraîné. Ce circuit d'une centaine de transistors est compact et efficace. On propose de continuer la recherche en attendant la production d'une puce reconfigurable qui pourrait catalyser l'utilisation générale des circuits de neurones à impulsions.

RÉFÉRENCES

J. Allman, *Evolving Brains*, New York : Scientific American Library/W. H. Freeman, 1998.

V. Braitenberg, *Vehikel. Experimente mit kybernetischen Wesen*, Allemagne : Rowohlt Taschenbuch, 1993.

F. Chénier, (communication personelle), 2005.

P. Churchland et J. Sejnowski, *The Computational Brain*, Cambridge : MIT Press, 1992.

T. Delbrück, "Bump circuits for computing similarity and dissimilarity of analog voltages", *Proc. IJCNN*, 1991, pp. 475-479.

R. Dorf et R. Bishop, *Modern Control Systems Ninth Edition* Upper Saddle River : Prentice Hall, 2000.

J. Dungen et A. Gherbi, "A Self-Normalizing, Statically Weighted, Layer-Level Building Block for Analog Pulsed Neural Networks", École Polytechnique de Montréal, Montréal, Canada, 2003.

J. Dungen, Laboratoire de réseaux de neurones, "Pulsing Self-Organizing Map", Août 2004, *http ://www.polymtl.ca/lrn/dungen/PulsingSOM/ Version1/JPulsingSOMApplet.jnlp.*

J. Dungen, Laboratoire de réseaux de neurones, "Pulsing Statically-Weighted Map", Octobre 2004, *http ://www.polymtl.ca/lrn/dungen/PulsingSOM/ Version2/JPulsingSOMApplet.jnlp.*

J. Dungen, Laboratoire de réseaux de neurones, "Simple Robot Brain", Novembre 2004, *http ://www.polymtl.ca/lrn/dungen/PulsingSOM/ Version3/JPulsingSOMApplet.jnlp.*

J. Dungen, Laboratoire de réseaux de neurones, "Create Your Own Pulsing Neural Network", Décembre 2004, *http ://www.polymtl.ca/lrn/dungen/PulsingSOM/ApplicationOne/JApplicationOne.jnlp.*

D. Floreanu, Y. Epars, J.C. Zufferey et C. Mattiussi, "Evolution of Spiking Neural Circuits in Autonomous Mobile Robots", *International Journal of Intelligent Systems*, vol. 21(9) pp. 1005-1024, 2006.

W. Gerstner et W. Kistler, *Spiking Neuron Models*, Cambridge : Cambridge University Press, 2002.

M. Glover, A. Hamilton et L. Smith, "Analogue VLSI Leaky Integrated-and-fire Neurons and their Use in a Sound Analysis System", *Analog Integrated Circuits and Signal Processing, Special Issue : Microelectronics for Bio-inspired Systems (Selected Papers from MicroNeuro'99 Conference)*, vol. 30(2), pp. 91-100, Février 2002.

S. Haykin, *Neural Networks a Comprehensive Foundation*, Upper Saddle River : Prentice Hall, 1999.

G. Indiveri, "Silicon Neurons and Silicon Synapses", notes de cours pour Computation in Neuromorphic analog VLSI Systems, Institute of Neuroinformatics, Université de Zürich, Hiver 2003.

G. Indiveri, "A Neuromorphic VLIS device for implementing 2D selective attention systems", *IEEE Transactions on Neural Networks*, vol. 12(6), pp. 1455-1463, Novembre 2001.

D. Johns et K. Martin, *Analog Integrated Circuit Design*, Toronto : John Wiley Sons, Inc., 1996.

T. Kohonen, "Self-organized formation of topologically correct feature maps", *Biological Cybernetics*, vol. 43, pp. 59-69, 1982.

T. Kohonen, *Self-Organization and Associative Memory*, Berlin : Springer-Verlag, 1984.

J. Koza et al., *Genetic Programming IV : Routine Human-Competitive Machine Intelligence*, New York : Springer, 2003.

J. Lazzaro et C. Mead, "A silicon model of auditory localization", *Neural Computation*, vol. 1, pp. 41-70, 1989.

M. Lewis, R. Etienne-Cummings, A. Cohen et M. Hartmann, "Toward Biomorphic Control Using Custom aVLSI CPG Chips", *Proceedings of the 2000 International conference on Robotics and Automation*, 2000, pp. 494-500.

Liu et al., *Analog VLSI : Circuits and Principles*, Cambridge : MIT Press, 2002.

W. Maass et G. Turan, "How fast can a threshold gate learn", *Computational Learning Theory and Natural Learning System : Constraints and Prospects*, Cambridge : MIT Press, 1994, pp. 281-414.

C. Mead (Ed.) et M. Ismail (Ed.), *Analog VLSI Implementation of Neural Systems*, Reading, MA : Addison-Wesley, 1989.

C. Mead et L. Conway, *Introduction to VLSI Systems*, Boston : Addison-Wesley, 1979.

D.S. Touretzky et D.A. Pomerleau, "What is hidden in the hidden layers ?" *Byte*, vol. 14, pp. 227-233, 1989.

ANNEXE I

CIRCUITS ALTERNATIVES

Circuits alternatives présentés dans le cours *Computation in Neuromorphic Analog Systems* à l'Institut de neuroinformatique à l'université de Zurich (Indiveri 2003).

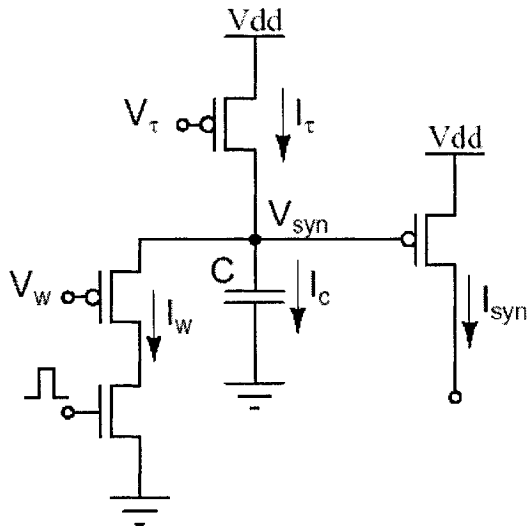

FIG. I.1 Log Domain Pulse Integrator

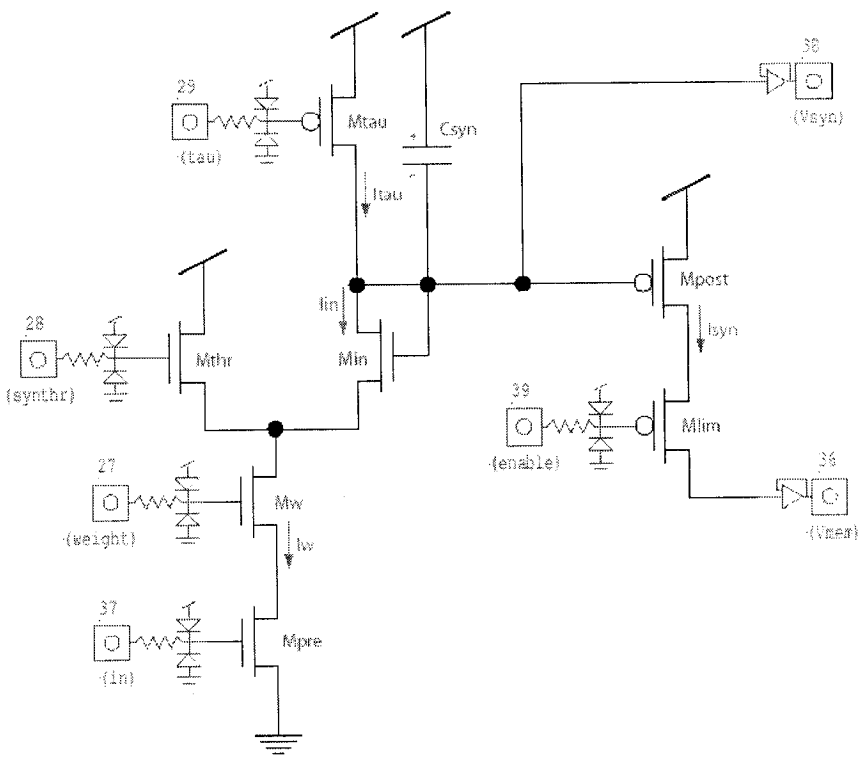

FIG. I.2 Differential Pair Integrator Synapse

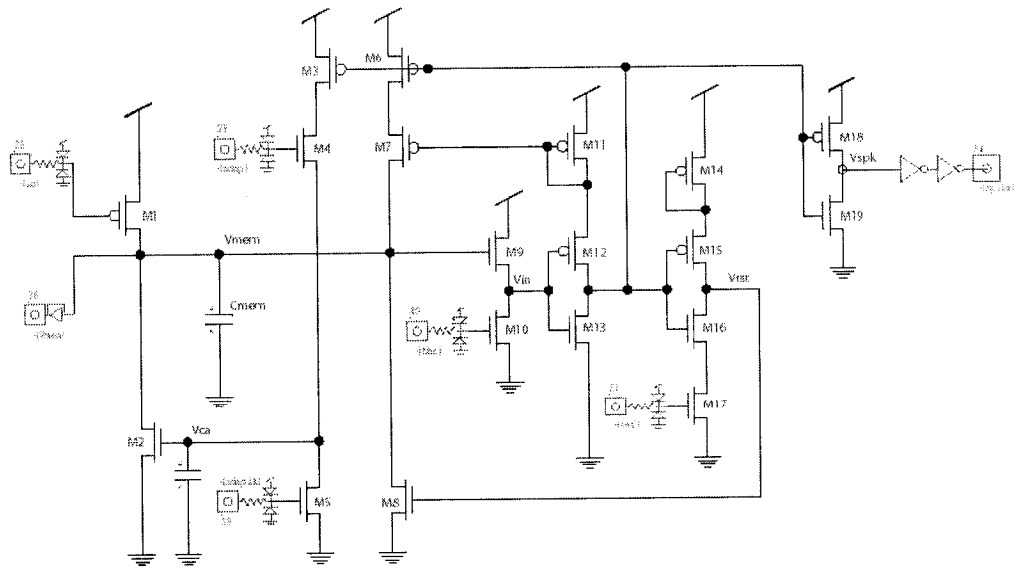

FIG. I.3 Low Power Integrate-and-Fire Neuron

ANNEXE II

CODE DE SIMULATION SPICE

Code Spice pour représenter une synapse :

```
* -- SUBCIRCUIT : JSynapseBlock -- *
.subckt JSynapseBlock 1 3 7 5

* -- Voltages -- *
Vdd 6 0 dc 1.8

* -- Transistors -- *
Minput 2 1 0 0 modn l=720n w=800n
Mweight 4 3 2 0 modn l=720n w=800n
Mgain 4 4 7 6 modp l=180n w=800n
Moutput 5 4 6 6 modp l=180n w=800n

* -- Capacitors -- *
Ccharge 4 0 8n

* -- Models and parameters -- *
.model modn nmos()
.model modp pmos()

.ends JSynapseBlock
* -- END OF SUBCIRCUIT -- *
```

Code Spice pour représenter un soma :

```
* -- SUBCIRCUIT : JSomaBlock -- *
.subckt JSomaBlock 2 1 5


* -- Voltages -- *
Vdd 7 0 dc 1.8
Vpw 3 0 dc 0.9

* -- Transistors -- *
Mreset 2 1 0 0 modn l=180n w=800n
Mpw 4 3 0 0 modn l=180n w=800n
Mbleed 2 5 4 0 modn l=180n w=800n
Minv1b 6 2 0 0 modn l=180n w=800n
Minv2b 5 6 0 0 modn l=180n w=800n
Minv1a 6 2 7 7 modp l=180n w=800n
Minv2a 5 6 7 7 modp l=180n w=800n

* -- Capacitors -- *
Ccharge 2 0 8n
Cfb 2 5 16n

* -- Models and parameters -- *
.model modn nmos()
.model modp pmos()

.ends
* -- END OF SUBCIRCUIT -- *
```

Code Spice pour simuler avec des tensions analogiques :

```
JNeuronTestVinput
```

```
.include JSynapseBlock.cir
.include JSomaBlock.cir

.tran 1u 60m

Xsynapse1 1 8 9 10 JSynapseBlock
Xsynapse2 2 8 9 11 JSynapseBlock
Xsynapse3 3 8 9 12 JSynapseBlock
Xsynapse4 4 8 9 13 JSynapseBlock
Xsynapse5 5 8 9 14 JSynapseBlock
Xsynapse6 6 8 9 15 JSynapseBlock
Xsynapse7 7 8 9 16 JSynapseBlock
Xsoma1 10 17 18 JSomaBlock
Xsoma2 11 17 19 JSomaBlock
Xsoma3 12 17 20 JSomaBlock
Xsoma4 13 17 21 JSomaBlock
Xsoma5 14 17 22 JSomaBlock
Xsoma6 15 17 23 JSomaBlock
Xsoma7 16 17 24 JSomaBlock

Vinput1 1 0 dc 0.7
Vinput2 2 0 dc 0.6
Vinput3 3 0 dc 0.5
Vinput4 4 0 dc 0.4
Vinput5 5 0 dc 0.3
Vinput6 6 0 dc 0.2
Vinput7 7 0 dc 0.15
Vweight 8 0 dc 0.7
Vfloor 9 0 dc 1.75
```

```
Vreset 17 0 pulse(0 1.8 0 500n 500n 2u 60m)

Rload1 18 0 10000k
Rload2 19 0 10000k
Rload3 20 0 10000k
Rload4 21 0 10000k
Rload5 22 0 10000k
Rload6 23 0 10000k
Rload7 24 0 10000k

.end
```

Code Spice pour simuler avec des impulsions :

```
JNeuronTestVinput

.include JSynapseBlock.cir
.include JSomaBlock.cir

.tran 1u 100m

Xsynapse1 1 8 9 10 JSynapseBlock
Xsynapse2 2 8 9 11 JSynapseBlock
Xsynapse3 3 8 9 12 JSynapseBlock
Xsynapse4 4 8 9 13 JSynapseBlock
Xsynapse5 5 8 9 14 JSynapseBlock
Xsynapse6 6 8 9 15 JSynapseBlock
Xsynapse7 7 8 9 16 JSynapseBlock
Xsoma1 10 17 18 JSomaBlock
Xsoma2 11 17 19 JSomaBlock
Xsoma3 12 17 20 JSomaBlock
```

```
Xsoma4 13 17 21 JSomaBlock
Xsoma5 14 17 22 JSomaBlock
Xsoma6 15 17 23 JSomaBlock
Xsoma7 16 17 24 JSomaBlock

Vinput1 1 0 pulse(0.015 1.65 1m 200u 200u 600u 2m)
Vinput2 2 0 pulse(0.015 1.65 1m 200u 200u 600u 3m)
Vinput3 3 0 pulse(0.015 1.65 1m 200u 200u 600u 4m)
Vinput4 4 0 pulse(0.015 1.65 1m 200u 200u 600u 6m)
Vinput5 5 0 pulse(0.015 1.65 1m 200u 200u 600u 8m)
Vinput6 6 0 pulse(0.015 1.65 1m 200u 200u 600u 10m)
Vinput7 7 0 pulse(0.015 1.65 1m 200u 200u 600u 12m)
Vweight 8 0 dc 0.5
Vfloor 9 0 dc 1.75
Vreset 17 0 pulse(0 1.8 0 500n 500n 2u 100m)

Rload1 18 0 10000k
Rload2 19 0 10000k
Rload3 20 0 10000k
Rload4 21 0 10000k
Rload5 22 0 10000k
Rload6 23 0 10000k
Rload7 24 0 10000k

.end
```